# FLTK Programming Manual

## 1 Preface

1.1 Preface

1.1.1 Organization

1.1.2 Conventions

1.1.3 Abbreviations

1.1.4 Copyrights and Trademarks

## 2 Introduction to FLTK

2.1 History of FLTK

2.2 Features

2.3 Licensing

2.4 What Does “FLTK” Mean?

2.5 Building and Installing FLTK with CMake

2.6 Building and Installing FLTK Under UNIX and macOS with make

2.7 Building FLTK Under Microsoft Windows

2.7.1 GNU toolsets (Cygwin or MinGW) hosted on Windows

2.7.2 Using the Visual C++ DLL Library

2.8 Internet Resources

2.9 Reporting Bugs

## 3 FLTK Basics

3.1 Writing Your First FLTK Program

3.1.1 Creating the Widgets

3.1.2 Creating Widget Hierarchies

3.1.3 Get/Set Methods

3.1.4 Redrawing After Changing Attributes

3.1.5 Labels

3.1.6 Showing the Window

3.1.7 The Main Event Loop

3.2 Naming Conventions

3.3 Header Files

3.4 Compiling Programs that Use FLTK

3.4.1 Compiling Programs with Standard Compilers

3.4.2 Compiling Programs with the 'fltk-config' Script

3.4.3 Compiling Multiple Source Files with 'fltk-config'

3.4.4 Compiling Programs with Makefiles

3.4.5 Compiling Programs with Microsoft Visual C++

## 4 Common Widgets and Attributes

4.1 Buttons

4.2 Text

4.3 Valuators

4.4 Groups

4.5 Setting the Size and Position of Widgets

4.6 Colors
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4.7 Box Types</td>
<td>22</td>
</tr>
<tr>
<td>1.4.7.1 Making Your Own Boxtypes</td>
<td>22</td>
</tr>
<tr>
<td>1.4.8 Labels and Label Types</td>
<td>23</td>
</tr>
<tr>
<td>1.4.9 Callbacks</td>
<td>28</td>
</tr>
<tr>
<td>1.4.10 When and Reason</td>
<td>29</td>
</tr>
<tr>
<td>1.4.11 Shortcuts</td>
<td>29</td>
</tr>
<tr>
<td>1.5 Coordinates and Layout Widgets</td>
<td>29</td>
</tr>
<tr>
<td>1.5.1 The Widget Coordinate System</td>
<td>29</td>
</tr>
<tr>
<td>1.5.2 Layout and Container Widgets</td>
<td>30</td>
</tr>
<tr>
<td>1.5.2.1 The Fl_Flex Layout Widget</td>
<td>30</td>
</tr>
<tr>
<td>1.5.2.2 The Fl_Grid Layout Widget</td>
<td>31</td>
</tr>
<tr>
<td>1.5.2.3 The Fl_Pack Layout Widget</td>
<td>32</td>
</tr>
<tr>
<td>1.5.2.4 The Fl_Scroll Container Widget</td>
<td>32</td>
</tr>
<tr>
<td>1.5.2.5 The Fl_Tabs Container Widget</td>
<td>33</td>
</tr>
<tr>
<td>1.5.2.6 The Fl_Tile Layout Widget</td>
<td>33</td>
</tr>
<tr>
<td>1.5.2.7 The Fl_Wizard Container Widget</td>
<td>33</td>
</tr>
<tr>
<td>1.6 How Does Resizing Work?</td>
<td>34</td>
</tr>
<tr>
<td>1.6.1 Resizing can be disabled</td>
<td>34</td>
</tr>
<tr>
<td>1.6.2 Resizing can be simple</td>
<td>34</td>
</tr>
<tr>
<td>1.6.3 Resizing can be complex</td>
<td>35</td>
</tr>
<tr>
<td>1.6.4 Practical examples</td>
<td>36</td>
</tr>
<tr>
<td>1.7 Designing a Simple Text Editor</td>
<td>39</td>
</tr>
<tr>
<td>1.7.1 Determining the Goals of the Text Editor</td>
<td>40</td>
</tr>
<tr>
<td>1.7.2 Chapter 1: A Minimal App</td>
<td>40</td>
</tr>
<tr>
<td>1.7.3 Chapter 2: Adding a Menu Bar</td>
<td>41</td>
</tr>
<tr>
<td>1.7.4 Chapter 3: Adding a Text Editor widget</td>
<td>42</td>
</tr>
<tr>
<td>1.7.5 Chapter 4: Reading and Writing Files</td>
<td>42</td>
</tr>
<tr>
<td>1.7.6 Chapter 5: Cut, Copy, and Paste</td>
<td>45</td>
</tr>
<tr>
<td>1.7.7 Chapter 6: Find and Find Next</td>
<td>45</td>
</tr>
<tr>
<td>1.7.8 Chapter 7: Replace and Replace Next</td>
<td>46</td>
</tr>
<tr>
<td>1.7.9 Chapter 8: Editor Features</td>
<td>48</td>
</tr>
<tr>
<td>1.7.10 Chapter 9: Split Editor</td>
<td>49</td>
</tr>
<tr>
<td>1.7.11 Chapter 10: Syntax Highlighting</td>
<td>50</td>
</tr>
<tr>
<td>1.8 Drawing Things in FLTK</td>
<td>53</td>
</tr>
<tr>
<td>1.8.1 When Can You Draw Things in FLTK?</td>
<td>53</td>
</tr>
<tr>
<td>1.8.2 What Units Do FLTK Functions Use?</td>
<td>53</td>
</tr>
<tr>
<td>1.8.3 Drawing Functions</td>
<td>54</td>
</tr>
<tr>
<td>1.8.3.1 Boxes</td>
<td>55</td>
</tr>
<tr>
<td>1.8.3.2 Clipping</td>
<td>56</td>
</tr>
<tr>
<td>1.8.3.3 Colors</td>
<td>57</td>
</tr>
<tr>
<td>1.8.3.4 Line Dashes and Thickness</td>
<td>59</td>
</tr>
<tr>
<td>1.8.3.5 Drawing Fast Shapes</td>
<td>60</td>
</tr>
</tbody>
</table>
1.16.10 Cursors .................................................. 154
1.16.11 FD "When" Conditions ................................. 154
1.16.12 Damage Masks .......................................... 155
1.17 GLUT Compatibility ........................................ 155
   1.17.1 Using the GLUT Compatibility Header File .... 155
   1.17.2 Known Problems ..................................... 155
   1.17.3 Mixing GLUT and FLTK Code ....................... 156
   1.17.4 class Fl_Glut_Window ................................ 157
      1.17.4.1 Class Hierarchy ................................. 157
      1.17.4.2 Include Files .................................. 157
      1.17.4.3 Description .................................... 157
      1.17.4.4 Members ....................................... 157
      1.17.4.5 Methods ...................................... 158
1.18 Forms Compatibility ...................................... 158
   1.18.1 Importing Forms Layout Files .................... 158
   1.18.2 Using the Compatibility Header File ............. 158
   1.18.3 Problems You Will Encounter .................... 159
   1.18.4 Additional Notes .................................. 161
1.19 Operating System Issues ................................ 162
   1.19.1 Accessing the OS Interfaces ...................... 162
   1.19.2 The Wayland/X11 hybrid library ................. 163
   1.19.3 The UNIX (X11) Interface ........................ 163
      1.19.3.1 Handling Other X Events ...................... 164
      1.19.3.2 Drawing using Xlib ........................... 165
      1.19.3.3 Changing the Display, Screen, or X Visual  165
      1.19.3.4 Using a Subclass of Fl_Window for Special X Stuff 167
      1.19.3.5 Setting the Icon of a Window ............... 169
      1.19.3.6 X Resources .................................. 169
      1.19.3.7 Display Scaling Factor ....................... 170
   1.19.4 The Windows Interface ............................. 170
      1.19.4.1 Using filenames with non-ASCII characters 170
      1.19.4.2 Responding to WM_QUIT ....................... 170
      1.19.4.3 Handling Other Windows API Messages ....... 171
      1.19.4.4 Drawing Things Using the Windows GDI .... 171
      1.19.4.5 HighDPI support ................................ 172
      1.19.4.6 Display Scaling Factor ....................... 172
      1.19.4.7 Setting the Icon of a Window ............... 172
      1.19.4.8 How to Not Get a MSDOS Console Window .... 172
      1.19.4.9 Known Windows Bugs and Problems .......... 173
   1.19.5 The Apple OS X Interface .......................... 173
      1.19.5.1 Setting the icon of an application ........ 175
      1.19.5.2 Drawing Things Using Quartz ................. 175
1.22.1.68 threads .......................................................... 193
1.22.1.69 tile .............................................................. 193
1.22.1.70 tiled_image .................................................. 193
1.22.1.71 tree ............................................................. 193
1.22.1.72 twowin .......................................................... 193
1.22.1.73 unitests ......................................................... 193
1.22.1.74 utf8 ............................................................. 193
1.22.1.75 valutors ......................................................... 194
1.22.1.76 windowfocus .................................................. 194
1.22.1.77 fluid ............................................................ 194
1.22.2 Example Applications: Images .................................. 194
  1.22.2.1 cairo_test .................................................... 194
  1.22.2.2 icon .......................................................... 194
  1.22.2.3 unitests ....................................................... 195
1.23 FAQ (Frequently Asked Questions) ..................................... 195
  1.23.1 Where do I start learning FLTK? ......................... 196
  1.23.2 How do I make a box with text? ......................... 196
  1.23.3 Can I use FLTK to make closed-source commercial applications? 196
  1.23.4 Hitting the 'Escape' key closes windows - how do I prevent this? 196

2 Fi_Terminal Technical Documentation ........................................ 199
  2.1 The Escape Codes Fi_Terminal Supports ....................... 199
  2.2 Useful Terminal Escape Code Documentation .................. 200
  2.3 Fi_Terminal Design Document .................................... 201

3 The gtk protocol ......................................................... 207
  3.1 Interfaces ........................................................ 207
  3.2 gtk_shell1 ........................................................ 207
    3.2.1 Description ................................................ 207
    3.2.2 API ......................................................... 207
  3.3 gtk_surface1 ..................................................... 207
    3.3.1 API ......................................................... 207

4 Development of the FLTK library ......................................... 209
  4.1 The Wayland backend for its developer ..................... 209
    4.1.1 Introduction to Wayland ................................ 209
    4.1.2 Building libfltk as a Wayland client .................. 210
    4.1.3 The hybrid Wayland/X11 platform ..................... 211
    4.1.4 Listeners .................................................. 211
    4.1.5 Opening a Wayland connection ......................... 212
    4.1.6 Wayland windows and surfaces ......................... 213
    4.1.7 Menu windows and other popups ...................... 214
    4.1.8 Fl_Wayland_Graphics_Driver and Fl_Cairo_Graphics_Driver 215
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.9 Wayland buffers</td>
<td>215</td>
</tr>
<tr>
<td>4.1.10 Buffer factories</td>
<td>217</td>
</tr>
<tr>
<td>4.1.11 Displays and HighDPI support</td>
<td>219</td>
</tr>
<tr>
<td>4.1.12 Mouse and trackpad handling</td>
<td>220</td>
</tr>
<tr>
<td>4.1.13 Wayland cursors</td>
<td>221</td>
</tr>
<tr>
<td>4.1.14 Keyboard support</td>
<td>222</td>
</tr>
<tr>
<td>4.1.15 Support of text input methods</td>
<td>222</td>
</tr>
<tr>
<td>4.1.16 Interface with libdecor</td>
<td>223</td>
</tr>
<tr>
<td>4.1.17 Copy/Paste/Drag-n-Drop</td>
<td>224</td>
</tr>
<tr>
<td>4.1.18 EGL as support for OpenGL</td>
<td>224</td>
</tr>
<tr>
<td>4.1.19 FLTK-defined, Wayland-specific types</td>
<td>225</td>
</tr>
<tr>
<td>4.1.20 Documentation resources</td>
<td>227</td>
</tr>
<tr>
<td>4.2 Developer info for bundled libs</td>
<td>227</td>
</tr>
<tr>
<td>4.2.1 Introduction</td>
<td>227</td>
</tr>
<tr>
<td>4.2.2 Current status</td>
<td>227</td>
</tr>
<tr>
<td>4.2.3 How to update the bundled libraries</td>
<td>228</td>
</tr>
<tr>
<td>4.2.4 zlib:</td>
<td>229</td>
</tr>
<tr>
<td>4.2.5 png:</td>
<td>229</td>
</tr>
<tr>
<td>4.2.6 jpeg:</td>
<td>230</td>
</tr>
<tr>
<td>4.2.7 nanosvg:</td>
<td>230</td>
</tr>
<tr>
<td>4.2.8 libdecor:</td>
<td>231</td>
</tr>
<tr>
<td>4.3 Developer Information</td>
<td>231</td>
</tr>
<tr>
<td>4.3.1 Non-ASCII Characters</td>
<td>233</td>
</tr>
<tr>
<td>4.3.2 Document Structure</td>
<td>233</td>
</tr>
<tr>
<td>4.3.3 Creating Links</td>
<td>234</td>
</tr>
<tr>
<td>4.3.4 Paragraph Layout</td>
<td>234</td>
</tr>
<tr>
<td>4.3.5 Navigation Elements</td>
<td>235</td>
</tr>
<tr>
<td>5 Todo List</td>
<td>237</td>
</tr>
<tr>
<td>6 Deprecated List</td>
<td>241</td>
</tr>
<tr>
<td>7 Topic Index</td>
<td>245</td>
</tr>
<tr>
<td>7.1 Topics</td>
<td>245</td>
</tr>
<tr>
<td>8 Hierarchical Index</td>
<td>247</td>
</tr>
<tr>
<td>8.1 Class Hierarchy</td>
<td>247</td>
</tr>
<tr>
<td>9 Class Index</td>
<td>251</td>
</tr>
<tr>
<td>9.1 Class List</td>
<td>251</td>
</tr>
<tr>
<td>10 File Index</td>
<td>259</td>
</tr>
<tr>
<td>10.1 File List</td>
<td>259</td>
</tr>
<tr>
<td>11 Topic Documentation</td>
<td>265</td>
</tr>
</tbody>
</table>
11.1 Callback Function Typedefs ............................................. 265
11.1.1 Detailed Description .................................................. 266
11.1.2 Typedef Documentation ................................................. 266
11.1.2.1 Fl_Event_Dispatch .................................................. 266
11.1.2.2 Fl_Timeout_Handler ................................................ 266
11.2 Windows handling functions ............................................. 266
11.2.1 Detailed Description ................................................... 267
11.2.2 Function Documentation ................................................ 267
11.2.2.1 default_atclose() .................................................. 267
11.2.2.2 first_window() [1/2] ................................................. 267
11.2.2.3 first_window() [2/2] ................................................. 267
11.2.2.4 grab() [1/2] ......................................................... 267
11.2.2.5 grab() [2/2] ......................................................... 267
11.2.2.6 modal() ............................................................. 267
11.2.2.7 next_window() ...................................................... 268
11.2.2.8 set_atclose() ....................................................... 268
11.2.3 Variable Documentation ............................................... 268
11.2.3.1 atclose ............................................................ 268
11.3 Events handling functions ................................................ 268
11.3.1 Detailed Description ................................................... 271
11.3.2 Function Documentation ................................................ 271
11.3.2.1 add_handler() ....................................................... 271
11.3.2.2 add_system_handler() .............................................. 271
11.3.2.3 belowmouse() [1/2] ................................................. 272
11.3.2.4 belowmouse() [2/2] ................................................. 272
11.3.2.5 callback_reason() ................................................ 272
11.3.2.6 compose() ......................................................... 272
11.3.2.7 compose_reset() .................................................. 272
11.3.2.8 disable_im() ....................................................... 273
11.3.2.9 enable_im() ....................................................... 273
11.3.2.10 event() .......................................................... 273
11.3.2.11 event_button() ................................................... 273
11.3.2.12 event_button1() .................................................. 273
11.3.2.13 event_button2() .................................................. 274
11.3.2.14 event_button3() .................................................. 274
11.3.2.15 event_buttons() .................................................. 274
11.3.2.16 event_clicks() [1/2] .............................................. 274
11.3.2.17 event_clicks() [2/2] .............................................. 274
11.3.2.18 event_clipboard() ................................................. 274
11.3.2.19 event_clipboard_type() ........................................... 275
11.3.2.20 event_dispatch() ................................................ 275
11.3.2.21 event_dx() ........................................................ 275
11.4.2.11 selection_to_clipboard() [2/2]  .................................................. 286
11.5 Screen functions  ................................................................. 287
   11.5.1 Detailed Description .................................................. 288
   11.5.2 Function Documentation ............................................. 288
      11.5.2.1 keyboard_screen_scaling() ....................................... 288
      11.5.2.2 screen_count() ...................................................... 288
      11.5.2.3 screen_dpi() .......................................................... 288
      11.5.2.4 screen_num() [1/2] .................................................. 289
      11.5.2.5 screen_num() [2/2] .................................................. 289
      11.5.2.6 screen_scale() ...................................................... 289
      11.5.2.7 screen_scaling_supported() ...................................... 289
      11.5.2.8 screen_work_area() [1/3] .......................................... 290
      11.5.2.9 screen_work_area() [2/3] .......................................... 290
      11.5.2.10 screen_work_area() [3/3] ......................................... 290
      11.5.2.11 screen_xywh() [1/4] ............................................... 291
      11.5.2.12 screen_xywh() [2/4] ............................................... 291
      11.5.2.13 screen_xywh() [3/4] ............................................... 291
      11.5.2.14 screen_xywh() [4/4] ............................................... 292
11.6 Color & Font functions ....................................................... 292
   11.6.1 Detailed Description .................................................. 294
   11.6.2 Function Documentation ............................................. 294
      11.6.2.1 fl_color() [1/3] ...................................................... 294
      11.6.2.2 fl_color() [2/3] ...................................................... 294
      11.6.2.3 fl_color() [3/3] ...................................................... 294
      11.6.2.4 fl_color_average() ................................................. 295
      11.6.2.5 fl_contrast() ......................................................... 295
      11.6.2.6 fl_contrast_function() ............................................ 296
      11.6.2.7 fl_contrast_level() [1/2] ......................................... 297
      11.6.2.8 fl_contrast_level() [2/2] ......................................... 297
      11.6.2.9 fl_contrast_mode() [1/2] .......................................... 298
      11.6.2.10 fl_contrast_mode() [2/2] ......................................... 298
      11.6.2.11 fl_font() [1/2] ....................................................... 299
      11.6.2.12 fl_font() [2/2] ....................................................... 299
      11.6.2.13 fl_height() [1/2] ..................................................... 299
      11.6.2.14 fl_height() [2/2] ..................................................... 299
      11.6.2.15 fl_latin1_to_local() .............................................. 299
      11.6.2.16 fl_lightness() ....................................................... 300
      11.6.2.17 fl_local_to_latin1() .............................................. 300
      11.6.2.18 fl_local_to_mac_roman() ........................................ 300
      11.6.2.19 fl_luminance() ...................................................... 301
      11.6.2.20 fl_mac_roman_to_local() ......................................... 301
      11.6.2.21 fl_show_colormap() ............................................... 302
11.7.3.22 fl_draw() [.3/4] ..................................................... 319
11.7.3.23 fl_draw() [.4/4] ..................................................... 319
11.7.3.24 fl_draw_arrow() .................................................... 319
11.7.3.25 fl_draw_box() ....................................................... 319
11.7.3.26 fl_draw_check() ..................................................... 320
11.7.3.27 fl_draw_circle() ..................................................... 320
11.7.3.28 fl_draw_pixmap() .................................................. 322
11.7.3.29 fl_draw_image() [.1/2] ............................................. 321
11.7.3.30 fl_draw_image() [.2/2] ............................................. 321
11.7.3.31 fl_draw_image_mono() [.1/2] .................................... 322
11.7.3.32 fl_draw_image_mono() [.2/2] .................................... 322
11.7.3.33 fl_draw_pixmap() [.2/2] .......................................... 323
11.7.3.34 fl_draw_radio() ...................................................... 323
11.7.3.35 fl_draw_symbol() ................................................... 323
11.7.3.36 fl_expand_text() ................................................... 324
11.7.3.37 fl_focus_rect() ...................................................... 324
11.7.3.38 fl_frame() ........................................................... 324
11.7.3.39 fl_frame2() ......................................................... 325
11.7.3.40 fl_gap() .............................................................. 325
11.7.3.41 fl_line_style() ..................................................... 325
11.7.3.42 fl_load_matrix() ................................................... 326
11.7.3.43 fl_measure() ........................................................ 326
11.7.3.44 fl_measure_pixmap() [.1/2] ...................................... 326
11.7.3.45 fl_measure_pixmap() [.2/2] ...................................... 327
11.7.3.46 fl_mult_matrix() ................................................... 327
11.7.3.47 fl_not_clipped() ................................................... 327
11.7.3.48 fl_old_shortcut() .................................................. 328
11.7.3.49 fl_overlay_clear() ................................................ 329
11.7.3.50 fl_overlay_rect() .................................................. 329
11.7.3.51 fl_override_scale() ................................................. 329
11.7.3.52 fl_pie() .............................................................. 330
11.7.3.53 fl_polygon() [.1/2] ............................................... 330
11.7.3.54 fl_polygon() [.2/2] ............................................... 330
11.7.3.55 fl_pop_clip() ........................................................ 331
11.7.3.56 fl_push_clip() ...................................................... 331
11.7.3.57 fl_push_matrix() ................................................... 331
11.7.3.58 fl_read_image() .................................................... 331
11.7.3.59 fl_rect() [.1/3] ...................................................... 332
11.7.3.60 fl_rect() [.2/3] ...................................................... 332
11.7.3.61 fl_rect() [.3/3] ...................................................... 332
11.7.3.62 fl_rectf() [.1/4] ..................................................... 332
11.7.3.63 fl_rectf() [.2/4] ..................................................... 332
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fl_rectf()</td>
<td>333</td>
</tr>
<tr>
<td>fl_rectf()</td>
<td>333</td>
</tr>
<tr>
<td>fl_scale()</td>
<td>334</td>
</tr>
<tr>
<td>fl_rotate()</td>
<td>334</td>
</tr>
<tr>
<td>fl_transform_dx()</td>
<td>337</td>
</tr>
<tr>
<td>fl_transform_dy()</td>
<td>337</td>
</tr>
<tr>
<td>fl_transform_x()</td>
<td>337</td>
</tr>
<tr>
<td>fl_transform_y()</td>
<td>337</td>
</tr>
<tr>
<td>fl_transformed_vertex()</td>
<td>338</td>
</tr>
<tr>
<td>fl_translate()</td>
<td>338</td>
</tr>
<tr>
<td>fl_vertex()</td>
<td>338</td>
</tr>
<tr>
<td>awake()</td>
<td>339</td>
</tr>
<tr>
<td>awake()</td>
<td>339</td>
</tr>
<tr>
<td>lock()</td>
<td>339</td>
</tr>
<tr>
<td>thread_message()</td>
<td>339</td>
</tr>
<tr>
<td>unlock()</td>
<td>340</td>
</tr>
<tr>
<td>clear_widget_pointer()</td>
<td>341</td>
</tr>
<tr>
<td>delete_widget()</td>
<td>341</td>
</tr>
<tr>
<td>do_widget_deletion()</td>
<td>341</td>
</tr>
<tr>
<td>release_widget_pointer()</td>
<td>342</td>
</tr>
<tr>
<td>watch_widget_pointer()</td>
<td>342</td>
</tr>
<tr>
<td>cairo_autolink_context()</td>
<td>343</td>
</tr>
</tbody>
</table>

11.8 Multithreading support functions

11.9 Safe widget deletion support functions

11.10 Cairo Support Functions and Classes
11.10.2.2 cairo_autolink_context() .................................................. 343
11.10.2.3 cairo_cc() ........................................................................ 344
11.10.2.4 cairo_flush() ..................................................................... 344
11.10.2.5 cairo_make_current() ............................................................. 344

11.11 Unicode and UTF-8 functions ......................................................... 345

11.11.1 Detailed Description .................................................................. 347

11.11.2 Macro Definition Documentation .............................................. 347

11.11.2.1 ERRORS_TO_CP1252 ............................................................... 347
11.11.2.2 ERRORS_TO_ISO8859_1 ......................................................... 347
11.11.2.3 STRICT RFC3629 ................................................................. 347

11.11.3 Function Documentation ............................................................ 347

11.11.3.1 fl_access() ......................................................................... 347
11.11.3.2 fl_chdir() ........................................................................... 348
11.11.3.3 fl_chmod() .......................................................................... 348
11.11.3.4 fl_close_fd() ....................................................................... 349
11.11.3.5 fl_fopen() ........................................................................... 349
11.11.3.6 fl_getcwd() .......................................................................... 349
11.11.3.7 fl_getenv() .......................................................................... 350
11.11.3.8 fl_make_path() ..................................................................... 350
11.11.3.9 fl_make_path_for_file() ............................................................ 350
11.11.3.10 fl_mkdir() .......................................................................... 351
11.11.3.11 fl_nonspacing() .................................................................. 351
11.11.3.12 fl_open() ............................................................................ 351
11.11.3.13 fl_open_ext() ..................................................................... 351
11.11.3.14 fl_putenv() ......................................................................... 352
11.11.3.15 fl_rename() ......................................................................... 352
11.11.3.16 fl_rmdir() ........................................................................... 353
11.11.3.17 fl_stat() ............................................................................. 353
11.11.3.18 fl_system() ......................................................................... 353
11.11.3.19 fl_ucs_to_Utf16() ................................................................. 354
11.11.3.20 fl_unlink() .......................................................................... 354
11.11.3.21 fl_utf8back() ...................................................................... 354
11.11.3.22 fl_utf8bytes() ..................................................................... 354
11.11.3.23 fl_utf8decode() ................................................................. 355
11.11.3.24 fl_utf8encode() ................................................................. 355
11.11.3.25 fl_utf8from_mb() ................................................................. 355
11.11.3.26 fl_utf8fromw() ................................................................. 356
11.11.3.27 fl_utf8fromwc() ................................................................. 356
11.11.3.28 fl_utf8fwd() ................................................................. 356
11.11.3.29 fl_utf8len() ....................................................................... 357
11.11.3.30 fl_utf8len1() ..................................................................... 357
11.11.3.31 fl_utf8locale() ................................................................. 357
<table>
<thead>
<tr>
<th>Section</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.11.3.32</td>
<td>fl_utf8strlen()</td>
<td>357</td>
</tr>
<tr>
<td>11.11.3.33</td>
<td>fl_utf8test()</td>
<td>357</td>
</tr>
<tr>
<td>11.11.3.34</td>
<td>fl_utf8to_mb()</td>
<td>358</td>
</tr>
<tr>
<td>11.11.3.35</td>
<td>fl_utf8toa()</td>
<td>358</td>
</tr>
<tr>
<td>11.11.3.36</td>
<td>fl_utf8tou16()</td>
<td>358</td>
</tr>
<tr>
<td>11.11.3.37</td>
<td>fl_utf8towc()</td>
<td>359</td>
</tr>
<tr>
<td>11.11.3.38</td>
<td>fl_utf_nb_char()</td>
<td>359</td>
</tr>
<tr>
<td>11.11.3.39</td>
<td>fl_utf_strcasecmp()</td>
<td>359</td>
</tr>
<tr>
<td>11.11.3.40</td>
<td>fl_utf_strncasecmp()</td>
<td>360</td>
</tr>
<tr>
<td>11.11.3.41</td>
<td>fl_utf_tolower()</td>
<td>360</td>
</tr>
<tr>
<td>11.11.3.42</td>
<td>fl_utf_toupper()</td>
<td>361</td>
</tr>
<tr>
<td>11.11.3.43</td>
<td>fl_wcwidth()</td>
<td>360</td>
</tr>
<tr>
<td>11.11.3.44</td>
<td>fl_wcwidth_()</td>
<td>361</td>
</tr>
</tbody>
</table>

11.12 String handling functions

11.12.1 Detailed Description

11.12.2 Function Documentation

11.12.2.1 fl_strdup()                  | 361  |

11.13 Mac OS X-specific symbols

11.13.1 Detailed Description

11.13.2 Function Documentation

11.13.2.1 fl_mac_set_about()          | 362  |
| 11.13.2.2 fl_open_callback()         | 362  |

11.13.3 Variable Documentation

11.13.3.1 fl_mac_os_version           | 362  |

11.14 Common Dialog Classes and Functions

11.14.1 Detailed Description

11.14.2 Function Documentation

11.14.2.1 fl_alert()                  | 364  |
| 11.14.2.2 fl_ask()                   | 365  |
| 11.14.2.3 fl_beep()                  | 365  |
| 11.14.2.4 fl_choice()                | 365  |
| 11.14.2.5 fl_choice_ni()             | 367  |
| 11.14.2.6 fl_color_chooser()         | 367  |
| 11.14.2.7 fl_color_chooser()         | 368  |
| 11.14.2.8 fl_dir_chooser()           | 369  |
| 11.14.2.9 fl_file_chooser()          | 370  |
| 11.14.2.10 fl_file_chooser_callback()| 371  |
| 11.14.2.11 fl_file_chooser_ok_label()| 371  |
| 11.14.2.12 fl_input()                | 371  |
| 11.14.2.13 fl_input()                | 371  |
| 11.14.2.14 fl_message()              | 372  |
| 11.14.2.15 fl_message_hotspot()      | 372  |

Generated by Doxygen
12.7.3.35 hide_all_windows() ......................................................... 409
12.7.3.36 is_scheme() ................................................................. 410
12.7.3.37 menu_linespacing() [1/2] .................................................. 410
12.7.3.38 menu_linespacing() [2/2] .................................................. 410
12.7.3.39 now() ........................................................................... 410
12.7.3.40 option() [1/2] ................................................................. 411
12.7.3.41 option() [2/2] ................................................................. 412
12.7.3.42 own colormap() ............................................................... 412
12.7.3.43 program_should_quit() [1/2] .............................................. 412
12.7.3.44 program_should_quit() [2/2] .............................................. 412
12.7.3.45 readqueue() ................................................................ 413
12.7.3.46 ready() ......................................................................... 413
12.7.3.47 release() ...................................................................... 413
12.7.3.48 reload_scheme() .............................................................. 413
12.7.3.49 remove_check() ............................................................... 414
12.7.3.50 remove_timeout() ............................................................ 414
12.7.3.51 repeat_timeout() ............................................................. 414
12.7.3.52 run() ............................................................................ 414
12.7.3.53 scheme() ...................................................................... 415
12.7.3.54 scrollbar_size() [1/2] ....................................................... 415
12.7.3.55 scrollbar_size() [2/2] ....................................................... 416
12.7.3.56 seconds_between() .......................................................... 416
12.7.3.57 seconds_since() .............................................................. 416
12.7.3.58 set_box_color() .............................................................. 417
12.7.3.59 set_idle() ..................................................................... 417
12.7.3.60 ticks_between() .............................................................. 417
12.7.3.61 ticks_since() ................................................................. 417
12.7.3.62 use_high_res_GL() [1/2] .................................................... 418
12.7.3.63 use_high_res_GL() [2/2] .................................................... 418
12.7.3.64 version() .................................................................... 418
12.7.3.65 visible_focus() [1/2] ....................................................... 418
12.7.3.66 visible_focus() [2/2] ....................................................... 418
12.7.3.67 visual() ...................................................................... 418
12.7.3.68 wait() [1/2] .................................................................. 419
12.7.3.69 wait() [2/2] .................................................................. 419

12.7.4 Member Data Documentation ............................................. 419
12.7.4.1 help .............................................................................. 419
12.7.4.2 idle ............................................................................. 419

12.8 Fl_Adjuster Class Reference .................................................. 420
12.8.1 Detailed Description ............................................................ 428
12.8.2 Constructor & Destructor Documentation ............................. 428
12.8.2.1 Fl_Adjuster() ................................................................. 428
12.8.3 Member Function Documentation ........................................ 428
  12.8.3.1 draw() ......................................................... 428
  12.8.3.2 handle() .................................................... 428
  12.8.3.3 soft() [1/2] ............................................... 429
  12.8.3.4 soft() [2/2] ............................................... 429
  12.8.3.5 value_damage() ........................................... 429

12.9 Fl_Anim_GIF_Image Class Reference ..................................... 429
  12.9.1 Detailed Description ............................................ 435
  12.9.2 Member Enumeration Documentation .................................. 435
    12.9.2.1 Flags ...................................................... 435
  12.9.3 Constructor & Destructor Documentation ............................ 435
    12.9.3.1 Fl_Anim_GIF_Image() [1/2] .................................. 435
    12.9.3.2 Fl_Anim_GIF_Image() [2/2] .................................. 436
    12.9.3.3 ~Fl_Anim_GIF_Image() ...................................... 436
  12.9.4 Member Function Documentation .................................... 436
    12.9.4.1 canvas() [1/2] ............................................. 436
    12.9.4.2 canvas() [2/2] ............................................. 437
    12.9.4.3 canvas_h() ................................................ 437
    12.9.4.4 canvas_w() ................................................ 437
    12.9.4.5 color_average() .......................................... 437
    12.9.4.6 copy() ..................................................... 438
    12.9.4.7 delay() [1/2] .............................................. 438
    12.9.4.8 delay() [2/2] .............................................. 438
    12.9.4.9 desaturate() ............................................... 438
    12.9.4.10 draw() ..................................................... 438
    12.9.4.11 frame() [1/2] ............................................. 439
    12.9.4.12 frame() [2/2] ............................................. 439
    12.9.4.13 frame_count() .......................................... 439
    12.9.4.14 frame_h() ................................................ 440
    12.9.4.15 frame_uncache() [1/2] .................................. 440
    12.9.4.16 frame_uncache() [2/2] .................................. 440
    12.9.4.17 frame_w() ................................................ 440
    12.9.4.18 frame_x() ................................................ 441
    12.9.4.19 frame_y() ................................................ 441
    12.9.4.20 frames() .................................................. 441
    12.9.4.21 image() [1/2] ............................................. 441
    12.9.4.22 image() [2/2] ............................................. 441
    12.9.4.23 is_animated() ........................................... 442
    12.9.4.24 load() ..................................................... 442
    12.9.4.25 name() .................................................... 442
    12.9.4.26 next() ..................................................... 442
    12.9.4.27 on_extension_data() .................................... 442
12.9.4.28 on_frame_data() .................................................. 443
12.9.4.29 playing() .......................................................... 443
12.9.4.30 resize() [1/2] ..................................................... 443
12.9.4.31 resize() [2/2] ..................................................... 443
12.9.4.32 speed() [1/2] ..................................................... 443
12.9.4.33 speed() [2/2] ..................................................... 443
12.9.4.34 start() ............................................................. 444
12.9.4.35 stop() .............................................................. 444
12.9.4.36 uncache() ........................................................ 444
12.9.4.37 valid() ............................................................. 444

12.9.5 Member Data Documentation ....................................... 444
12.9.5.1 loop ................................................................. 444
12.9.5.2 min_delay .......................................................... 444

12.10 Fl_Bitmap Class Reference ........................................... 444
12.10.1 Detailed Description ............................................... 447
12.10.2 Constructor & Destructor Documentation ....................... 447
12.10.2.1 Fl_Bitmap() [1/4] ............................................... 447
12.10.2.2 Fl_Bitmap() [2/4] ............................................... 447
12.10.2.3 Fl_Bitmap() [3/4] ............................................... 448
12.10.2.4 Fl_Bitmap() [4/4] ............................................... 448
12.10.3 Member Function Documentation ................................ 448
12.10.3.1 copy() ............................................................ 448
12.10.3.2 draw() ............................................................ 449
12.10.3.3 label() [1/2] ..................................................... 449
12.10.3.4 label() [2/2] ..................................................... 449
12.10.3.5 uncache() ........................................................ 450

12.11 Fl_BMP_Image Class Reference ..................................... 450
12.11.1 Detailed Description ............................................... 453
12.11.2 Constructor & Destructor Documentation ....................... 453
12.11.2.1 Fl_BMP_Image() [1/2] ........................................... 453
12.11.2.2 Fl_BMP_Image() [2/2] ........................................... 453

12.12 Fl_Box Class Reference ............................................... 454
12.12.1 Detailed Description ............................................... 460
12.12.2 Constructor & Destructor Documentation ....................... 461
12.12.2.1 Fl_Box() .......................................................... 461
12.12.3 Member Function Documentation ................................ 461
12.12.3.1 draw() ............................................................ 461
12.12.3.2 handle() .......................................................... 461

12.13 Fl_Browser Class Reference ......................................... 462
12.13.1 Detailed Description ............................................... 475
12.13.2 Constructor & Destructor Documentation ....................... 475
12.13.2.1 Fl_Browser() ..................................................... 475

Generated by Doxygen
12.13.3 Member Function Documentation .............................................. 476
  12.13.3.1 _remove() ............................................................... 476
  12.13.3.2 add() ..................................................................... 476
  12.13.3.3 bottomline() .............................................................. 476
  12.13.3.4 clear() ..................................................................... 476
  12.13.3.5 column_char() [1/2] .................................................... 477
  12.13.3.6 column_char() [2/2] .................................................... 477
  12.13.3.7 column_widths() [1/2] .................................................. 477
  12.13.3.8 column_widths() [2/2] .................................................. 477
  12.13.3.9 data() [1/2] ................................................................. 478
  12.13.3.10 data() [2/2] ............................................................... 479
  12.13.3.11 display() ................................................................ 479
  12.13.3.12 displayed() ............................................................... 479
  12.13.3.13 find_line() ............................................................... 480
  12.13.3.14 format_char() [1/2] .................................................... 480
  12.13.3.15 format_char() [2/2] .................................................... 481
  12.13.3.16 full_height() ............................................................. 481
  12.13.3.17 hide() [1/2] ............................................................... 481
  12.13.3.18 hide() [2/2] ............................................................... 481
  12.13.3.19 icon() [1/2] ............................................................... 482
  12.13.3.20 icon() [2/2] ............................................................... 482
  12.13.3.21 incr_height() ............................................................ 482
  12.13.3.22 insert() [1/2] ............................................................. 482
  12.13.3.23 insert() [2/2] ............................................................. 483
  12.13.3.24 item_at() ................................................................. 483
  12.13.3.25 item_draw() ............................................................. 483
  12.13.3.26 item_first() .............................................................. 484
  12.13.3.27 item_height() .......................................................... 484
  12.13.3.28 item_last() ............................................................... 484
  12.13.3.29 item_next() .............................................................. 485
  12.13.3.30 item_prev() .............................................................. 485
  12.13.3.31 item_select() ........................................................... 485
  12.13.3.32 item_selected() ........................................................ 486
  12.13.3.33 item_swap() ............................................................. 486
  12.13.3.34 item_text() .............................................................. 486
  12.13.3.35 item_width() ........................................................... 487
  12.13.3.36 lineno() ................................................................. 487
  12.13.3.37 lineposition() ......................................................... 487
  12.13.3.38 load() ................................................................. 488
  12.13.3.39 make_visible() ....................................................... 488
  12.13.3.40 middleline() ............................................................ 488
  12.13.3.41 move() ................................................................. 488
12.13.3.42 remove() .................................................. 490
12.13.3.43 remove_icon() ........................................... 490
12.13.3.44 select() ................................................... 490
12.13.3.45 selected() ................................................ 491
12.13.3.46 show() [1/2] ................................................ 491
12.13.3.47 show() [2/2] ............................................... 491
12.13.3.48 size() ..................................................... 491
12.13.3.49 swap() [1/2] ............................................... 491
12.13.3.50 swap() [2/2] ............................................... 492
12.13.3.51 text() [1/2] ............................................... 492
12.13.3.52 text() [2/2] ............................................... 492
12.13.3.53 textsize() .................................................. 492
12.13.3.54 topline() [1/2] ........................................... 493
12.13.3.55 topline() [2/2] ........................................... 493
12.13.3.56 value() [1/2] ............................................... 493
12.13.3.57 value() [2/2] ............................................... 493
12.13.3.58 visible() .................................................. 493

12.14 Fl_Browser_ Class Reference .................................... 494
  12.14.1 Detailed Description ........................................ 505
  12.14.2 Member Enumeration Documentation ......................... 505
    12.14.2.1 anonymous enum ...................................... 505
  12.14.3 Constructor & Destructor Documentation ..................... 506
    12.14.3.1 Fl_Browser_() ....................................... 506
  12.14.4 Member Function Documentation ............................ 506
    12.14.4.1 bbox() ............................................... 506
    12.14.4.2 deleting() ............................................ 506
    12.14.4.3 deselect() .......................................... 506
    12.14.4.4 display() ............................................. 507
    12.14.4.5 displayed() .......................................... 507
    12.14.4.6 draw() ............................................... 507
    12.14.4.7 find_item() .......................................... 507
    12.14.4.8 full_height() ....................................... 508
    12.14.4.9 full_width() ........................................ 508
    12.14.4.10 handle() ............................................. 508
    12.14.4.11 has_scrollbar() .................................... 508
    12.14.4.14 incr_height() ....................................... 509
    12.14.4.15 inserting() .......................................... 509
    12.14.4.16 item_at() ............................................ 509
    12.14.4.17 item_draw() ......................................... 510
    12.14.4.18 item_first() ........................................ 510

Generated by Doxygen
12.14.4.19 item_height() .................................................. 510
12.14.4.20 item_last() .................................................. 510
12.14.4.21 item_next() .................................................. 511
12.14.4.22 item_prev() .................................................. 511
12.14.4.23 item_quick_height() ....................................... 511
12.14.4.24 item_select() ............................................... 511
12.14.4.25 item_selected() ............................................ 511
12.14.4.26 item_swap() ................................................ 512
12.14.4.27 item_text() ................................................ 512
12.14.4.28 item_width() ............................................... 512
12.14.4.29 leftedge() .................................................. 513
12.14.4.30 linespacing() [1/2] ........................................ 513
12.14.4.31 linespacing() [2/2] ........................................ 513
12.14.4.32 new_list() .................................................. 513
12.14.4.33 position() [1/2] ............................................ 513
12.14.4.34 position() [2/2] ............................................ 513
12.14.4.35 redraw_line() .............................................. 514
12.14.4.36 redraw_lines() ............................................. 514
12.14.4.37 replacing() ................................................ 514
12.14.4.38 resize() ..................................................... 514
12.14.4.39 scrollbar_left() .......................................... 514
12.14.4.40 scrollbar_right() ........................................ 515
12.14.4.41 scrollbar_size() [1/2] .................................... 515
12.14.4.43 scrollbar_width() [1/2] ................................ 515
12.14.4.44 scrollbar_width() [2/2] ................................ 515
12.14.4.45 select() ..................................................... 516
12.14.4.46 select_only() .............................................. 516
12.14.4.47 selection() ................................................ 516
12.14.4.48 sort() ....................................................... 516
12.14.4.49 swapping() ................................................ 517
12.14.4.50 textfont() .................................................. 517
12.14.4.51 vposition() [1/2] ....................................... 517
12.14.5 Member Data Documentation .................................. 517
12.14.5.1 hscrollbar .................................................. 517
12.14.5.2 scrollbar .................................................... 518
12.15 Fl_Button Class Reference ........................................... 518
12.15.1 Detailed Description ........................................... 525
12.15.2 Constructor & Destructor Documentation ....................... 525
12.15.2.1 Fl_Button() ............................................... 525
12.15.3 Member Function Documentation ................................ 526
12.15.3.1 clear() ......................................................... 526
12.15.3.2 compact() [1/2] ........................................... 526
12.15.3.3 compact() [2/2] ........................................... 526
12.15.3.4 down_box() [1/2] ......................................... 527
12.15.3.5 down_box() [2/2] ......................................... 527
12.15.3.6 draw() ......................................................... 527
12.15.3.7 handle() ....................................................... 528
12.15.3.8 set() .......................................................... 528
12.15.3.9 shortcut() [1/2] ............................................ 528
12.15.3.10 shortcut() [2/2] ........................................... 529
12.15.3.11 value() ....................................................... 529

12.16 Fl_Cairo_State Class Reference ........................................ 529
  12.16.1 Detailed Description ........................................ 530
  12.16.2 Member Function Documentation ................................ 530
  12.16.2.1 cc() ....................................................... 530

12.17 Fl_Cairo_Window Class Reference ..................................... 530
  12.17.1 Detailed Description ......................................... 542
  12.17.2 Member Function Documentation ................................ 543
    12.17.2.1 draw() ................................................. 543
    12.17.2.2 set_draw_cb() .......................................... 543

12.18 Fl_Callback_User_Data Class Reference ................................ 543
  12.18.1 Detailed Description ........................................ 544

12.19 Fl_Chart Class Reference ............................................ 544
  12.19.1 Detailed Description ........................................ 551
  12.19.2 Constructor & Destructor Documentation ...................... 552
    12.19.2.1 Fl_Chart() ............................................. 552
  12.19.3 Member Function Documentation ................................ 552
    12.19.3.1 add() ................................................... 552
    12.19.3.2 autosize() [1/2] ...................................... 553
    12.19.3.3 autosize() [2/2] ...................................... 553
    12.19.3.4 bounds() [1/2] ....................................... 553
    12.19.3.5 bounds() [2/2] ....................................... 553
    12.19.3.6 draw() ................................................... 553
    12.19.3.7 draw_barchart() ...................................... 554
    12.19.3.8 draw_horbarchart() ................................. 554
    12.19.3.9 draw_linechart() ..................................... 555
    12.19.3.10 draw_piechart() ..................................... 555
    12.19.3.11 insert() ............................................... 555
    12.19.3.12 maxsize() ............................................ 556
    12.19.3.13 replace() ............................................ 556
    12.19.3.14 size() ................................................ 556

12.20 FL_CHART_ENTRY Struct Reference ...................................... 557
12.25.1 Detailed Description ........................................... 608
12.25.2 Constructor & Destructor Documentation .......................... 608
   12.25.2.1 Fl_Clock_Output() ........................................ 608
12.25.3 Member Function Documentation ................................ 609
   12.25.3.1 draw() [1/2] ........................................... 609
   12.25.3.2 draw() [2/2] ........................................... 609
   12.25.3.3 hour() ................................................ 609
   12.25.3.4 minute() .............................................. 609
   12.25.3.5 second() .............................................. 609
   12.25.3.6 shadow() [1/2] ......................................... 610
   12.25.3.7 shadow() [2/2] ......................................... 610
   12.25.3.8 value() [1/3] .......................................... 610
   12.25.3.9 value() [2/3] .......................................... 610
   12.25.3.10 value() [3/3] ......................................... 611
12.26 Fl_Color_Chooser Class Reference .................................. 611
   12.26.1 Detailed Description ........................................ 620
   12.26.2 Constructor & Destructor Documentation ........................ 620
      12.26.2.1 Fl_Color_Chooser() .................................... 620
   12.26.3 Member Function Documentation ................................ 621
      12.26.3.1 b() .................................................. 621
      12.26.3.2 g() .................................................. 621
      12.26.3.3 handle() ............................................. 621
      12.26.3.4 hsv() .............................................. 621
      12.26.3.5 hsv2rgb() .......................................... 622
      12.26.3.6 hue() ................................................ 622
      12.26.3.7 mode() [1/2] ......................................... 622
      12.26.3.8 mode() [2/2] ......................................... 622
      12.26.3.9 r() .................................................. 622
      12.26.3.10 rgb() .............................................. 622
      12.26.3.11 rgb2hsv() ......................................... 623
      12.26.3.12 saturation() ....................................... 623
      12.26.3.13 value() ........................................... 623
12.27 Fl_Copy_Surface Class Reference .................................. 623
   12.27.1 Detailed Description ........................................ 625
   12.27.2 Constructor & Destructor Documentation ........................ 626
      12.27.2.1 Fl_Copy_Surface() ................................... 626
   12.27.3 Member Function Documentation ................................ 626
      12.27.3.1 is_current() ......................................... 626
      12.27.3.2 origin() [1/2] ....................................... 626
      12.27.3.3 origin() [2/2] ....................................... 626
      12.27.3.4 printable_rect() .................................... 626
      12.27.3.5 set_current() ....................................... 627
12.56.3.17 margin() ................................. 838
12.56.3.18 margin() ................................. 839
12.56.3.19 need_layout() ......................... 839
12.56.3.20 on_remove() ......................... 840
12.56.3.21 resize() ................................. 840
12.56.3.22 row_gap() ........................................ 840
12.56.3.23 row_gap() ........................................ 840
12.56.3.24 row_height() ............................... 840
12.56.3.25 row_height() ............................... 841
12.56.3.26 row_weight() ............................... 841
12.56.3.27 row_weight() ............................... 841
12.56.3.28 show_grid() ............................... 841
12.56.3.29 show_grid() ............................... 842
12.56.3.30 widget() ........................................ 842
12.56.3.31 widget() ........................................ 843

12.57 Fl_Group Class Reference ....................... 843

12.57.1 Detailed Description .......................... 852

12.57.2 Constructor & Destructor Documentation ......... 852
  12.57.2.1 Fl_Group() ................................. 852
  12.57.2.2 ~Fl_Group() ................................. 852

12.57.3 Member Function Documentation .................. 853
  12.57.3.1 array() ........................................ 853
  12.57.3.2 as_group() ....................................... 853
  12.57.3.3 as_group() ....................................... 853
  12.57.3.4 begin() ......................................... 853
  12.57.3.5 bounds() ........................................ 853
  12.57.3.6 child() ......................................... 854
  12.57.3.7 clear() ......................................... 854
  12.57.3.8 clip_children() ................................ 854
  12.57.3.9 clip_children() ................................ 854
  12.57.3.10 current() ...................................... 855
  12.57.3.11 current() ...................................... 855
  12.57.3.12 delete_child() ......................... 855
  12.57.3.13 draw() ......................................... 856
  12.57.3.14 draw_child() .............................. 856
  12.57.3.15 draw_children() ............................ 856
  12.57.3.16 end() ......................................... 856
  12.57.3.17 find() ......................................... 856
  12.57.3.18 focus() ....................................... 856
  12.57.3.19 handle() ...................................... 856
  12.57.3.20 init_sizes() .............................. 857
  12.57.3.21 insert() ....................................... 857
12.57.3.22 insert() .................................................. 858
12.57.3.23 on_insert() .............................................. 858
12.57.3.24 on_move() ................................................. 858
12.57.3.25 on_remove() ............................................... 859
12.57.3.26 remove() .................................................. 859
12.57.3.27 remove() .................................................. 859
12.57.3.28 remove() .................................................. 859
12.57.3.29 resizable() ............................................... 859
12.57.3.30 resizable() ............................................... 860
12.57.3.31 resizable() ............................................... 860
12.57.3.32 resize() ................................................... 861
12.57.3.33 sizes() ..................................................... 861
12.57.3.34 update_child() ........................................... 862

12.58 Fl_Help_Block Struct Reference .................................. 862
12.59 Fl_Help_Dialog Class Reference .................................. 863
12.59.1 Detailed Description ........................................... 864
12.59.2 Member Function Documentation ................................ 864
12.59.2.1 load() .................................................... 864
12.59.2.2 show() .................................................... 864
12.59.2.3 textsize() ................................................. 864
12.59.2.4 value() .................................................... 865
12.59.2.5 value() .................................................... 865

12.60 Fl_Help_Font_Stack Struct Reference ................................. 865
12.61 Fl_Help_Font_Style Struct Reference ................................. 865
12.61.1 Detailed Description ........................................... 866
12.62 Fl_Help_Link Struct Reference .................................... 866
12.62.1 Detailed Description ........................................... 866
12.63 Fl_Help_Target Struct Reference .................................. 866
12.63.1 Detailed Description ........................................... 867
12.64 Fl_Help_View Class Reference ..................................... 867
12.64.1 Detailed Description ........................................... 876
12.64.2 Constructor & Destructor Documentation ............................ 876
12.64.2.1 ~Fl_Help_View() ........................................... 876
12.64.2.3 Member Function Documentation ................................ 878
12.64.3.1 draw() .................................................... 878
12.64.3.2 find() .................................................... 878
12.64.3.3 handler() ................................................. 878
12.64.3.4 leftline() ................................................. 879
12.64.3.5 link() .................................................... 879
12.64.3.6 load() .................................................... 879
12.64.3.7 resize() .................................................. 879
12.64.3.8 scrollbar_size() ......................................... 880
12.64.3.9 scrollbar_size() [2/2] ................................................. 880
12.64.3.10 topline() [1/2] ................................................... 880
12.64.3.11 topline() [2/2] ................................................... 880
12.64.3.12 value() ............................................................... 881

12.65 Fl_Hold_Browser Class Reference ........................................ 881
  12.65.1 Detailed Description ............................................. 894
  12.65.2 Constructor & Destructor Documentation ......................... 894
    12.65.2.1 Fl_Hold_Browser() ......................................... 894

12.66 Fl_Hor_Fill_Slider Class Reference ..................................... 894

12.67 Fl_Hor_Nice_Slider Class Reference .................................... 903
  12.67.1 Detailed Description ............................................. 911

12.68 Fl_Hor_Slider Class Reference ......................................... 911
  12.68.1 Detailed Description ............................................. 919

12.69 Fl_Hor_Value_Slider Class Reference ................................... 919

12.70 Fl_ICO_Image Class Reference .......................................... 928
  12.70.1 Detailed Description ............................................. 931
  12.70.2 Constructor & Destructor Documentation ............................ 931
    12.70.2.1 Fl_ICO_Image() ............................................... 931

12.71 Fl_Image Class Reference ............................................. 932
  12.71.1 Detailed Description ............................................. 934
  12.71.2 Constructor & Destructor Documentation ............................ 934
    12.71.2.1 Fl_Image() .................................................. 934
  12.71.3 Member Function Documentation .................................... 934
    12.71.3.1 as_shared_image() . ......................................... 934
    12.71.3.2 color_average() . ............................................ 935
    12.71.3.3 copy() [1/2] .................................................. 935
    12.71.3.4 copy() [2/2] .................................................. 935
    12.71.3.5 count() ....................................................... 936
    12.71.3.6 d() .......................................................... 936
    12.71.3.7 data() [1/2] .................................................. 936
    12.71.3.8 data() [2/2] .................................................. 936
    12.71.3.9 desaturate() .................................................. 937
    12.71.3.10 draw() [1/2] ................................................ 937
    12.71.3.11 draw() [2/2] ................................................ 937
    12.71.3.12 draw_empty() ................................................. 937
    12.71.3.13 draw_scaled() .............................................. 937
    12.71.3.14 fail() ....................................................... 938
    12.71.3.15 h() [1/2] ..................................................... 938
    12.71.3.16 h() [2/2] ..................................................... 938
    12.71.3.17 inactive() ................................................... 939
    12.71.3.18 label() [1/2] ................................................ 939
    12.71.3.19 label() [2/2] ................................................ 939
12.75.3.1 append() .............................................. 970
12.75.3.2 apply_undo() ......................................... 970
12.75.3.3 can_redo() ........................................... 970
12.75.3.4 can_undo() ........................................... 971
12.75.3.5 copy() .................................................. 971
12.75.3.6 copy_cuts() ........................................... 971
12.75.3.7 cursor_color() ....................................... 971
12.75.3.8 cursor_color() ....................................... 971
12.75.3.9 cut() ................................................... 972
12.75.3.10 cut() ................................................... 972
12.75.3.11 cut() ................................................... 972
12.75.3.12 drawtext() ........................................... 972
12.75.3.13 drawtext() ........................................... 973
12.75.3.14 dvalue() ............................................. 973
12.75.3.15 handle_mouse() ...................................... 973
12.75.3.16 handletext() ........................................ 974
12.75.3.17 index() ............................................... 974
12.75.3.18 input_type() ......................................... 974
12.75.3.19 input_type() ......................................... 974
12.75.3.20 insert() ............................................... 974
12.75.3.21 insert_position() .................................. 975
12.75.3.22 insert_position() .................................. 975
12.75.3.23 insert_position() ................................ 975
12.75.3.24 ivalue() .............................................. 976
12.75.3.25 line_end() ........................................... 976
12.75.3.26 line_start() ......................................... 976
12.75.3.27 mark() ............................................... 976
12.75.3.28 mark() ............................................... 977
12.75.3.29 maximum_size() .................................... 977
12.75.3.30 maximum_size() .................................... 977
12.75.3.31 position() .......................................... 977
12.75.3.32 position() .......................................... 977
12.75.3.33 position() .......................................... 978
12.75.3.34 readonly() .......................................... 978
12.75.3.35 readonly() .......................................... 978
12.75.3.36 redo() ................................................ 978
12.75.3.37 replace() ............................................ 978
12.75.3.38 resize() ............................................. 979
12.75.3.39 shortcut() .......................................... 979
12.75.3.40 shortcut() .......................................... 979
12.75.3.41 size() ................................................. 980
12.75.3.42 size() ................................................. 980
12.83.16 insert() .......................................................... 1048
12.83.17 item_pathname() ............................................... 1049
12.83.18 menu() [1/2] .................................................. 1049
12.83.19 menu() [2/2] .................................................. 1050
12.83.20 menu_box() [1/2] ............................................ 1050
12.83.21 menu_box() [2/2] ............................................ 1050
12.83.22 menu_end() ................................................... 1050
12.83.23 mode() [1/2] .................................................. 1051
12.83.24 mode() [2/2] .................................................. 1051
12.83.25 mvalue() ...................................................... 1051
12.83.26 picked() ....................................................... 1051
12.83.27 prev_mvalue() ............................................... 1051
12.83.28 remove() ...................................................... 1052
12.83.29 replace() ...................................................... 1052
12.83.30 size() ......................................................... 1052
12.83.31 test_shortcut() .............................................. 1052
12.83.32 text() [1/2] ................................................. 1052
12.83.33 text() [2/2] ................................................. 1052
12.83.34 textcolor() ................................................... 1053
12.83.35 textfont() [1/2] ............................................ 1053
12.83.36 textfont() [2/2] ............................................ 1053
12.83.37 textsize() [1/2] ............................................ 1053
12.83.38 textsize() [2/2] ............................................ 1053
12.83.39 value() [1/3] ................................................ 1053
12.83.40 value() [2/3] ................................................ 1053
12.83.41 value() [3/3] ................................................ 1053

12.84 Fl_Menu_Bar Class Reference .................................... 1054
  12.84.1 Detailed Description ....................................... 1063
  12.84.2 Constructor & Destructor Documentation .................. 1063
    12.84.2.1 Fl_Menu_Bar() ....................................... 1063
  12.84.3 Member Function Documentation .......................... 1063
    12.84.3.1 draw() ................................................ 1063
    12.84.3.2 handle() ............................................. 1064
    12.84.3.3 update() ............................................. 1064

12.85 Fl_Menu_Button Class Reference ................................ 1064
  12.85.1 Detailed Description ...................................... 1073
  12.85.2 Member Enumeration Documentation ....................... 1074
    12.85.2.1 popup_buttons ....................................... 1074
  12.85.3 Constructor & Destructor Documentation .................. 1075
    12.85.3.1 Fl_Menu_Button() .................................... 1075
  12.85.4 Member Function Documentation .......................... 1075
    12.85.4.1 draw() ................................................ 1075
12.105.4.3 Fl_Preferences() [3/8] ........................................ 1235
12.105.4.4 Fl_Preferences() [4/8] ........................................ 1235
12.105.4.5 Fl_Preferences() [5/8] ........................................ 1235
12.105.4.6 Fl_Preferences() [6/8] ........................................ 1236
12.105.4.7 Fl_Preferences() [7/8] ........................................ 1236
12.105.4.8 ~Fl_Preferences() ........................................ 1236
12.105.4.9 Fl_Preferences() ........................................ 1236

12.105.5 Member Function Documentation ........................................ 1237
12.105.5.1 delete_entry() ........................................ 1237
12.105.5.2 delete_group() ........................................ 1237
12.105.5.3 dirty() ........................................ 1237
12.105.5.4 entries() ........................................ 1237
12.105.5.5 entry() ........................................ 1237
12.105.5.6 entry_exists() ........................................ 1237
12.105.5.7 file_access() [1/2] ........................................ 1238
12.105.5.8 file_access() [2/2] ........................................ 1238
12.105.5.9 filename() [1/2] ........................................ 1239
12.105.5.10 filename() [2/2] ........................................ 1239
12.105.5.11 flush() ........................................ 1239
12.105.5.12 get() [1/8] ........................................ 1240
12.105.5.13 get() [2/8] ........................................ 1240
12.105.5.14 get() [3/8] ........................................ 1240
12.105.5.15 get() [4/8] ........................................ 1241
12.105.5.16 get() [5/8] ........................................ 1241
12.105.5.17 get() [6/8] ........................................ 1241
12.105.5.18 get() [7/8] ........................................ 1242
12.105.5.19 get() [8/8] ........................................ 1242
12.105.5.20 get_userdata_path() ........................................ 1243
12.105.5.21 group() ........................................ 1243
12.105.5.22 group_exists() ........................................ 1244
12.105.5.23 groups() ........................................ 1244
12.105.5.24 new_UUID() ........................................ 1244
12.105.5.25 set() [1/7] ........................................ 1244
12.105.5.26 set() [2/7] ........................................ 1245
12.105.5.27 set() [3/7] ........................................ 1245
12.105.5.28 set() [4/7] ........................................ 1245
12.105.5.29 set() [5/7] ........................................ 1247
12.105.5.30 set() [6/7] ........................................ 1247
12.105.5.31 set() [7/7] ........................................ 1247
12.105.5.32 size() ........................................ 1248

12.105.6 Member Data Documentation ........................................ 1248
12.105.6.1 CORE_READ_OK ........................................ 1248
12.105.6.2 CORE_WRITE_OK ........................................ 1248
12.105.6.3 NONE ............................................... 1248

12.106 FL_Printer Class Reference .................................... 1248
  12.106.1 Detailed Description ........................................ 1252
  12.106.2 Member Function Documentation ............................. 1253
    12.106.2.1 begin_job() ........................................ 1253
    12.106.2.2 begin_page() ........................................ 1253
    12.106.2.3 end_job() ............................................ 1254
    12.106.2.4 end_page() ........................................... 1254
    12.106.2.5 is_current() .......................................... 1254
    12.106.2.6 margins() ............................................. 1254
    12.106.2.7 origin() ............................................... 1255
    12.106.2.8 origin() ............................................... 1255
    12.106.2.9 printable_rect() .................................... 1255
    12.106.2.10 rotate() ............................................. 1256
    12.106.2.11 scale() ............................................... 1256
    12.106.2.12 set_current() ....................................... 1256
    12.106.2.13 translate() ......................................... 1256
    12.106.2.14 untranslate() ....................................... 1256

12.107 FL_Progress Class Reference ................................. 1257
  12.107.1 Detailed Description ....................................... 1263
  12.107.2 Constructor & Destructor Documentation .................... 1263
    12.107.2.1 FL_Progress() ......................................... 1263
  12.107.3 Member Function Documentation ................................ 1264
    12.107.3.1 draw() ............................................... 1264
    12.107.3.2 maximum() ........................................... 1264
    12.107.3.3 maximum() ........................................... 1264
    12.107.3.4 minimum() ............................................ 1264
    12.107.3.5 minimum() ............................................ 1264
    12.107.3.6 value() ................................................ 1264
    12.107.3.7 value() ................................................ 1264

12.108 FL_Radio_Button Class Reference ............................ 1265
  12.108.1 Constructor & Destructor Documentation .................... 1272
    12.108.1.1 FL_Radio_Button() .................................... 1272

12.109 FL_Radio_Light_Button Class Reference ...................... 1272

12.110 FL_Radio_Round_Button Class Reference ...................... 1280
  12.110.1 Constructor & Destructor Documentation .................... 1287
    12.110.1.1 FL_Radio_Round_Button() ................................ 1287

12.111 FL_Rect Class Reference ...................................... 1288
  12.111.1 Detailed Description ....................................... 1289
  12.111.2 Constructor & Destructor Documentation .................... 1289
    12.111.2.1 FL_Rect() ............................................. 1289

Generated by Doxygen
12.140.4.17 resize() ........................................ 1562
12.140.4.18 tab_align() [1/2] .............................. 1562
12.140.4.19 tab_align() [2/2] .............................. 1562
12.140.4.20 tab_height() .................................. 1562
12.140.4.21 tab_positions() ................................ 1563
12.140.4.22 value() [1/2] .................................. 1563
12.140.4.23 value() [2/2] .................................. 1564
12.140.4.24 which() ........................................ 1564
12.140.5 Member Data Documentation ...................... 1564
  12.140.5.1 overflow_type .................................. 1564
  12.140.5.2 tab_count ..................................... 1564
  12.140.5.3 tab_flags ...................................... 1565
  12.140.5.4 tab_pos ........................................ 1565
  12.140.5.5 tab_width ...................................... 1565
12.141 Fl_Terminal Class Reference ............................. 1565
  12.141.1 Detailed Description .............................. 1580
  12.141.2 Fl_Terminal ....................................... 1580
    12.141.2.1 Examples .................................... 1581
    12.141.2.2 Writing To Terminal From Applications ......... 1582
    12.141.2.3 Text Attributes ................................ 1582
    12.141.2.4 Text and Background Colors ................. 1582
    12.141.2.5 Features ...................................... 1584
    12.141.2.6 Margins ...................................... 1584
    12.141.2.7 Caveats ....................................... 1585
  12.141.3 Member Enumeration Documentation .............. 1585
    12.141.3.1 Attrib ....................................... 1585
    12.141.3.2 CharFlags .................................... 1586
    12.141.3.3 OutFlags ..................................... 1586
    12.141.3.4 RedrawStyle .................................. 1586
  12.141.4 Constructor & Destructor Documentation ......... 1587
    12.141.4.1 Fl_Terminal() [1/2] ........................... 1587
    12.141.4.2 Fl_Terminal() [2/2] ........................... 1587
    12.141.4.3 ~Fl_Terminal() ................................ 1587
  12.141.5 Member Function Documentation .................. 1588
    12.141.5.1 ansi() [1/2] .................................. 1588
    12.141.5.2 ansi() [2/2] .................................. 1588
    12.141.5.3 append() ...................................... 1588
    12.141.5.4 append_ascii() ................................ 1589
    12.141.5.5 append_utf8() ................................ 1589
    12.141.5.6 box() ......................................... 1589
    12.141.5.7 clear() [1/2] .................................. 1589
    12.141.5.8 clear() [2/2] .................................. 1589
12.143.2 Member Enumeration Documentation ............................................. 1639
  12.143.2.1 anonymous enum ............................................................... 1639
  12.143.2.2 anonymous enum ............................................................... 1639
  12.143.2.3 anonymous enum ............................................................... 1639
12.143.3 Constructor & Destructor Documentation ......................................... 1640
  12.143.3.1 Fl_Text_Display() ............................................................... 1640
  12.143.3.2 ~Fl_Text_Display() ............................................................. 1640
12.143.4 Member Function Documentation .................................................. 1640
  12.143.4.1 absolute_top_line_number() .................................................. 1640
  12.143.4.2 buffer() [1/3] ..................................................................... 1640
  12.143.4.3 buffer() [2/3] ..................................................................... 1641
  12.143.4.4 buffer() [3/3] ..................................................................... 1641
  12.143.4.5 buffer_modified_cb() .............................................................. 1641
  12.143.4.6 buffer_predelete_cb() .............................................................. 1642
  12.143.4.7 calc_last_char() .................................................................. 1642
  12.143.4.8 calc_line_starts() ................................................................. 1642
  12.143.4.9 clear_rect() ....................................................................... 1642
  12.143.4.10 col_to_x() ....................................................................... 1643
  12.143.4.11 count_lines() ..................................................................... 1643
  12.143.4.12 cursor_color() [1/2] .............................................................. 1643
  12.143.4.13 cursor_color() [2/2] .............................................................. 1643
  12.143.4.14 cursor_style() ................................................................... 1644
  12.143.4.15 display_insert() .................................................................. 1644
  12.143.4.16 draw() ............................................................................. 1644
  12.143.4.17 draw_cursor() .................................................................... 1644
  12.143.4.18 draw_line_numbers() ............................................................ 1645
  12.143.4.19 draw_range() ..................................................................... 1645
  12.143.4.20 draw_string() .................................................................... 1645
  12.143.4.21 draw_text() ....................................................................... 1645
  12.143.4.22 draw_vline() ..................................................................... 1646
  12.143.4.23 empty_vlines() ................................................................... 1646
  12.143.4.24 extend_range_for_styles() .................................................... 1646
  12.143.4.25 find_line_end() ................................................................... 1647
  12.143.4.26 find_wrap_range() ................................................................ 1647
  12.143.4.27 find_x() ........................................................................... 1647
  12.143.4.28 get_absolute_top_line_number() .............................................. 1648
  12.143.4.29 grammar_underline_color() [1/2] .............................................. 1648
  12.143.4.30 grammar_underline_color() [2/2] .............................................. 1648
  12.143.4.31 handle() ........................................................................... 1648
  12.143.4.32 handle_rmb() ..................................................................... 1648
  12.143.4.33 handle_vline() .................................................................... 1649
  12.143.4.34 highlight_data() .................................................................. 1649
12.143.4.77 secondary_selection_color() [1/2] ........................................................................ 1661
12.143.4.78 secondary_selection_color() [2/2] ........................................................................ 1661
12.143.4.79 shortcut() [1/2] ........................................................................................................ 1661
12.143.4.80 shortcut() [2/2] ........................................................................................................ 1661
12.143.4.81 show_cursor() ............................................................................................................... 1662
12.143.4.82 show_insert_position() .................................................................................................. 1662
12.143.4.83 skip_lines() ...................................................................................................................... 1662
12.143.4.84 spelling_underline_color() [1/2] ...................................................................................... 1662
12.143.4.85 spelling_underline_color() [2/2] ...................................................................................... 1662
12.143.4.86 string_width() ................................................................................................................ 1662
12.143.4.87 style_buffer() .................................................................................................................. 1663
12.143.4.88 textcolor() [1/2] .............................................................................................................. 1663
12.143.4.89 textcolor() [2/2] .............................................................................................................. 1663
12.143.4.90 textfont() [1/2] ................................................................................................................ 1663
12.143.4.91 textfont() [2/2] ................................................................................................................ 1663
12.143.4.92 textsize() [1/2] .................................................................................................................. 1664
12.143.4.93 textsize() [2/2] .................................................................................................................. 1664
12.143.4.94 update_h_scrollbar() ...................................................................................................... 1664
12.143.4.95 update_line_starts() ....................................................................................................... 1664
12.143.4.96 update_v_scrollbar() ...................................................................................................... 1664
12.143.4.97 vline_length() .................................................................................................................. 1665
12.143.4.98 word_end() ....................................................................................................................... 1665
12.143.4.99 word_start() ..................................................................................................................... 1665
12.143.4.100 wrap_mode() .................................................................................................................. 1665
12.143.4.101 wrap_uses_character() .................................................................................................. 1666
12.143.4.102 wrapped_column() ....................................................................................................... 1666
12.143.4.103 wrapped_line_counter() ............................................................................................... 1667
12.143.4.104 wrapped_row() ............................................................................................................. 1667
12.143.4.105 x_to_col() ....................................................................................................................... 1668
12.143.4.106 xy_to_position() ............................................................................................................ 1668
12.143.4.107 xy_to_rowcol() .............................................................................................................. 1668

12.144 Fl_Text_Editor Class Reference ................................................................................................. 1669

12.144.1 Detailed Description .............................................................................................................. 1686

12.144.2 Member Function Documentation ......................................................................................... 1686
12.144.2.1 add_key_binding() .............................................................................................................. 1686
12.144.2.2 handle() ............................................................................................................................... 1686
12.144.2.3 insert_mode() [1/2] ............................................................................................................ 1686
12.144.2.4 insert_mode() [2/2] ............................................................................................................ 1686
12.144.2.5 kf_backspace() .................................................................................................................. 1686
12.144.2.6 kf_c_s_move() ...................................................................................................................... 1687
12.144.2.7 kf_copy() ............................................................................................................................. 1687
12.144.2.8 kf_ctrl_move() ................................................................................................................... 1687
12.146.2 Constructor & Destructor Documentation ........................................... 1706
  12.146.2.1 Fl_Tile() ............................................................... 1706

12.146.3 Member Function Documentation .................................................. 1707
  12.146.3.1 cursor() ......................................................................... 1707
  12.146.3.2 drag_intersection() ......................................................... 1707
  12.146.3.3 handle() ......................................................................... 1707
  12.146.3.4 init_size_range() ............................................................ 1708
  12.146.3.5 move_intersection() ......................................................... 1708
  12.146.3.6 on_insert() ...................................................................... 1708
  12.146.3.7 on_move() ........................................................................ 1708
  12.146.3.8 on_remove() ...................................................................... 1708
  12.146.3.9 position() ........................................................................ 1709
  12.146.3.10 request_grow_b() ............................................................... 1709
  12.146.3.11 request_grow_l() ............................................................... 1709
  12.146.3.12 request_grow_r() ............................................................... 1709
  12.146.3.13 request_grow_t() ............................................................... 1710
  12.146.3.14 request_shrink_b() ............................................................. 1710
  12.146.3.15 request_shrink_l() ............................................................. 1710
  12.146.3.16 request_shrink_r() ............................................................. 1711
  12.146.3.17 request_shrink_t() ............................................................. 1711
  12.146.3.18 resize() ......................................................................... 1711
  12.146.3.19 set_cursor() ................................................................. 1712
  12.146.3.20 size_range() [1/2] ............................................................. 1712
  12.146.3.21 size_range() [2/2] ............................................................. 1712

12.147 Fl_Tiled_Image Class Reference ....................................................... 1713
  12.147.1 Detailed Description .................................................................... 1715
  12.147.2 Constructor & Destructor Documentation .................................. 1715
    12.147.2.1 Fl_Tiled_Image() ............................................................ 1715
  12.147.3 Member Function Documentation .......................................... 1716
    12.147.3.1 color_average() .............................................................. 1716
    12.147.3.2 copy() ..................................................................... 1716
    12.147.3.3 desaturate() ................................................................. 1716
    12.147.3.4 draw() ..................................................................... 1717

12.148 Fl_Timeout Class Reference ............................................................. 1717
  12.148.1 Detailed Description .................................................................... 1718
  12.148.2 Member Function Documentation .......................................... 1719
    12.148.2.1 add_timeout() .............................................................. 1719
    12.148.2.2 current() ..................................................................... 1720
    12.148.2.3 elapse_timeouts() ......................................................... 1720
    12.148.2.4 get() ..................................................................... 1720
    12.148.2.5 has_timeout() .............................................................. 1721
    12.148.2.6 insert() ................................................................. 1721
12.152 Fl_Tree Class Reference .................................................. 1745
  12.152.1 Detailed Description .................................................. 1759
  12.152.2 Member Function Documentation .................................... 1763
    12.152.2.1 add() [1/2] ...................................................... 1763
    12.152.2.2 add() [2/2] ...................................................... 1764
    12.152.2.3 calc_dimensions() ............................................. 1764
    12.152.2.4 calc_tree() .................................................... 1764
    12.152.2.5 callback_item() [1/2] ....................................... 1765
    12.152.2.6 callback_item() [2/2] ....................................... 1765
    12.152.2.7 callback_reason() [1/2] .................................... 1765
    12.152.2.8 callback_reason() [2/2] .................................... 1765
    12.152.2.9 clear() .......................................................... 1765
    12.152.2.10 clear_children() .......................................... 1766
    12.152.2.11 close() [1/2] ................................................ 1766
    12.152.2.12 close() [2/2] ................................................ 1766
    12.152.2.13 closeicon() [1/2] .......................................... 1767
    12.152.2.14 closeicon() [2/2] .......................................... 1767
    12.152.2.15 connectorstyle() ........................................... 1767
    12.152.2.16 deselect() [1/2] ........................................... 1767
    12.152.2.17 deselect() [2/2] ........................................... 1768
    12.152.2.18 deselect_all() ............................................. 1768
    12.152.2.19 display() .................................................... 1769
    12.152.2.20 displayed() .................................................. 1769
    12.152.2.21 draw() ........................................................ 1769
    12.152.2.22 extend_selection() ........................................ 1769
    12.152.2.23 extend_selection_dir() ................................... 1770
    12.152.2.24 find_clicked() ............................................. 1770
    12.152.2.25 find_item() ................................................ 1771
    12.152.2.26 first() ........................................................ 1771
    12.152.2.27 first_selected_item() .................................... 1772
    12.152.2.28 first_visible() ............................................ 1772
    12.152.2.29 first_visible_item() ..................................... 1772
    12.152.2.30 get_selected_items() ..................................... 1772
12.152.2.31 handle() .......................................................... 1773
12.152.2.32 hposition() [1/2] ........................................ 1773
12.152.2.33 hposition() [2/2] ........................................ 1773
12.152.2.34 insert() .......................................................... 1774
12.152.2.35 insert_above() .............................................. 1774
12.152.2.36 is_close() [1/2] .............................................. 1775
12.152.2.37 is_close() [2/2] .............................................. 1775
12.152.2.38 is_hscroll_visible() ...................................... 1775
12.152.2.39 is_open() [1/2] .............................................. 1776
12.152.2.40 is_open() [2/2] .............................................. 1776
12.152.2.41 is_scrollbar() ............................................... 1776
12.152.2.42 is_selected() [1/2] ......................................... 1777
12.152.2.43 is_selected() [2/2] ......................................... 1777
12.152.2.44 is_vscroll_visible() ...................................... 1777
12.152.2.45 item_clicked() [1/2] ...................................... 1777
12.152.2.46 item_clicked() [2/2] ...................................... 1778
12.152.2.47 item_draw_mode() [1/3] ................................. 1778
12.152.2.48 item_draw_mode() [2/3] ................................. 1778
12.152.2.49 item_draw_mode() [3/3] ................................. 1778
12.152.2.50 item_labelbgcolor() [1/2] ......................... 1778
12.152.2.51 item_labelbgcolor() [2/2] ......................... 1778
12.152.2.52 item_labelfgcolor() .................................... 1779
12.152.2.53 item_labelfont() ........................................ 1779
12.152.2.54 item_labelsize() ........................................ 1779
12.152.2.55 item_pathname() ......................................... 1779
12.152.2.56 item_reselect_mode() [1/2] ....................... 1779
12.152.2.57 item_reselect_mode() [2/2] ....................... 1780
12.152.2.58 last() ......................................................... 1780
12.152.2.59 last_selected_item() .................................. 1780
12.152.2.60 last_visible() ............................................ 1780
12.152.2.61 last_visible_item() ..................................... 1781
12.152.2.62 load() ......................................................... 1781
12.152.2.63 next() .......................................................... 1781
12.152.2.64 next_item() .................................................. 1781
12.152.2.65 next_selected_item() ................................... 1782
12.152.2.66 next_visible_item() ..................................... 1783
12.152.2.67 open() [1/2] ................................................. 1784
12.152.2.68 open() [2/2] ................................................. 1784
12.152.2.69 open_toggle() ............................................... 1785
12.152.2.70 openicon() [1/2] ......................................... 1785
12.152.2.71 openicon() [2/2] ......................................... 1785
12.152.2.72 prev() .......................................................... 1785
12.152.2.73 recalc_tree() ........................................ 1786
12.152.2.74 remove() .............................................. 1786
12.152.2.75 resize() ................................................. 1786
12.152.2.76 root() .................................................. 1787
12.152.2.77 root_label() ............................................ 1787
12.152.2.78 scrollbar_size() [1/2] ................................. 1787
12.152.2.79 scrollbar_size() [2/2] ................................. 1787
12.152.2.80 select() [1/2] ........................................... 1787
12.152.2.81 select() [2/2] ........................................... 1788
12.152.2.82 select_all() ............................................. 1788
12.152.2.83 select_only() .......................................... 1789
12.152.2.84 select_toggle() ......................................... 1789
12.152.2.85 selectbox() [1/2] ....................................... 1790
12.152.2.86 selectbox() [2/2] ....................................... 1790
12.152.2.87 selectmode() [1/2] ..................................... 1790
12.152.2.88 selectmode() [2/2] ..................................... 1790
12.152.2.89 set_item_focus() ........................................ 1790
12.152.2.90 show_item() [1/2] ....................................... 1790
12.152.2.91 show_item() [2/2] ....................................... 1791
12.152.2.92 show_item_bottom() .................................... 1791
12.152.2.93 show_item_middle() ..................................... 1791
12.152.2.94 show_item_top() ......................................... 1791
12.152.2.95 show_self() ............................................. 1792
12.152.2.96 showcollapse() [1/2] ................................. 1792
12.152.2.97 showcollapse() [2/2] ................................. 1792
12.152.2.98 showroot() .............................................. 1792
12.152.2.99 sortorder() .............................................. 1792
12.152.2.100 usericon() [1/2] ...................................... 1793
12.152.2.101 usericon() [2/2] ...................................... 1793
12.152.2.102 vposition() [1/2] ..................................... 1793
12.152.2.103 vposition() [2/2] ..................................... 1793

12.153 Fl_Tree_Item Class Reference ..................................... 1793
12.153.1 Detailed Description ....................................... 1798
12.153.2 Constructor & Destructor Documentation ................... 1799
12.153.2.1 Fl_Tree_Item() [1/2] .................................... 1799
12.153.2.2 Fl_Tree_Item() [2/2] .................................... 1799
12.153.3 Member Function Documentation ...................................... 1799
12.153.3.1 activate() .............................................. 1799
12.153.3.2 add() [1/4] .............................................. 1799
12.153.3.3 add() [2/4] .............................................. 1800
12.153.3.5 add() [4/4] .............................................. 1800

Generated by Doxygen
12.153.3.6 calc_item_height() ........................................ 1800
12.153.3.7 child() ........................................ 1801
12.153.3.8 deactivate() ........................................ 1801
12.153.3.9 deparent() ........................................ 1801
12.153.3.10 depth() ........................................ 1801
12.153.3.11 deselect_all() ........................................ 1801
12.153.3.12 draw() ........................................ 1802
12.153.3.13 draw_horizontal_connector() ........................................ 1802
12.153.3.14 draw_item_content() ........................................ 1802
12.153.3.15 draw_vertical_connector() ........................................ 1803
12.153.3.16 drawbgcolor() ........................................ 1804
12.153.3.17 drawfgcolor() ........................................ 1804
12.153.3.18 find_child() [1/2] ........................................ 1804
12.153.3.19 find_child() [2/2] ........................................ 1804
12.153.3.20 find_child_item() [1/2] ........................................ 1804
12.153.3.21 find_child_item() [2/2] ........................................ 1805
12.153.3.22 find_clicked() ........................................ 1805
12.153.3.23 find_item() ........................................ 1805
12.153.3.24 hide_widgets() ........................................ 1805
12.153.3.25 insert() ........................................ 1805
12.153.3.26 insert_above() ........................................ 1806
12.153.3.27 label() ........................................ 1806
12.153.3.28 label_h() ........................................ 1806
12.153.3.29 label_w() ........................................ 1806
12.153.3.30 label_x() ........................................ 1806
12.153.3.31 label_y() ........................................ 1807
12.153.3.32 labelbgcolor() [1/2] ........................................ 1807
12.153.3.33 labelbgcolor() [2/2] ........................................ 1807
12.153.3.34 move() [1/2] ........................................ 1807
12.153.3.35 move() [2/2] ........................................ 1808
12.153.3.36 move_above() ........................................ 1808
12.153.3.37 move_below() ........................................ 1808
12.153.3.38 move_into() ........................................ 1808
12.153.3.39 next() ........................................ 1809
12.153.3.40 next_displayed() ........................................ 1809
12.153.3.41 next_sibling() ........................................ 1809
12.153.3.42 next_visible() ........................................ 1809
12.153.3.43 parent() ........................................ 1809
12.153.3.44 prefs() ........................................ 1810
12.153.3.45 prev() ........................................ 1810
12.153.3.46 prev_displayed() ........................................ 1810
12.153.3.47 prev_sibling() ........................................ 1810
12.153.3.48 prev_visible() .......................................................... 1810
12.153.3.49 recalc_tree() .......................................................... 1810
12.153.3.50 remove_child() [1/2] ................................................. 1811
12.153.3.51 remove_child() [2/2] ................................................. 1811
12.153.3.52 reparent() ................................................................. 1811
12.153.3.53 replace() ................................................................. 1811
12.153.3.54 replace_child() ....................................................... 1812
12.153.3.55 select() ................................................................. 1812
12.153.3.56 select_all() ............................................................ 1812
12.153.3.57 show_self() ............................................................. 1812
12.153.3.58 show_widgets() ....................................................... 1813
12.153.3.59 swap_children() [1/2] ............................................... 1813
12.153.3.60 swap_children() [2/2] ............................................... 1813
12.153.3.61 tree() [1/2] ............................................................. 1813
12.153.3.62 tree() [2/2] ............................................................. 1813
12.153.3.63 update_prev_next() .................................................. 1814
12.153.3.64 userdeicon() ........................................................... 1814
12.153.3.65 userdeicon() [2/2] .................................................... 1814
12.153.3.66 usericon() ............................................................... 1814
12.153.3.67 visible_r() ............................................................. 1814
12.154 Fl_Tree_Item_Array Class Reference ..................................... 1815
12.154.1 Detailed Description ..................................................... 1816
12.154.2 Constructor & Destructor Documentation ............................ 1816
12.154.2.1 Fl_Tree_Item_Array() .................................................. 1816
12.154.3 Member Function Documentation ...................................... 1816
12.154.3.1 add() ................................................................. 1816
12.154.3.2 clear() ................................................................. 1816
12.154.3.3 deparent() .............................................................. 1816
12.154.3.4 insert() ................................................................. 1816
12.154.3.5 manage_item_destroy() ............................................. 1817
12.154.3.6 move() ................................................................. 1817
12.154.3.7 remove() [1/2] .......................................................... 1817
12.154.3.8 remove() [2/2] .......................................................... 1817
12.154.3.9 reparent() .............................................................. 1817
12.154.3.10 replace() .............................................................. 1817
12.155 Fl_Tree_Prefs Class Reference ........................................... 1818
12.155.1 Detailed Description ..................................................... 1820
12.155.2 Member Function Documentation ...................................... 1821
12.155.2.1 closedeicon() ........................................................ 1821
12.155.2.2 closeicon() ............................................................ 1821
12.155.2.3 item_draw_mode() .................................................... 1821
12.155.2.4 item_labelbgcolor() [1/2] .......................................... 1821
12.155.2.5 item_labelbgcolor() [2/2] ........................................... 1821
12.155.2.6 marginbottom() .................................................. 1821
12.155.2.7 openendicon() .................................................. 1821
12.155.2.8 openicon() [1/2] .................................................. 1821
12.155.2.9 openicon() [2/2] .................................................. 1822
12.155.2.10 selectmode() ................................................... 1822
12.155.2.11 showcollapse() .................................................. 1822
12.155.2.12 showroot() ....................................................... 1822
12.155.2.13 sortorder() ....................................................... 1822
12.155.2.14 userdeicon() .................................................... 1822

12.156 Fl_Valuator Class Reference ........................................... 1823
12.156.1 Detailed Description ............................................. 1830
12.156.2 Constructor & Destructor Documentation .................. 1831
12.156.2.1 Fl_Valuator() .................................................... 1831
12.156.3 Member Function Documentation .............................. 1831
12.156.3.1 format() ......................................................... 1831
12.156.3.2 increment() ....................................................... 1831
12.156.3.3 maximum() [1/2] ................................................. 1832
12.156.3.4 maximum() [2/2] ................................................. 1832
12.156.3.5 minimum() [1/2] ................................................. 1832
12.156.3.6 minimum() [2/2] ................................................. 1832
12.156.3.7 precision() ....................................................... 1832
12.156.3.8 range() ........................................................... 1832
12.156.3.9 round() ........................................................... 1833
12.156.3.10 step() ........................................................... 1833
12.156.3.11 value() [1/2] ..................................................... 1833
12.156.3.12 value() [2/2] ..................................................... 1833
12.156.3.13 value_damage() ................................................. 1833

12.157 Fl_Value_Input Class Reference .................................. 1833
12.157.1 Detailed Description ............................................. 1841
12.157.2 Constructor & Destructor Documentation .................. 1842
12.157.2.1 Fl_Value_Input() .................................................. 1842
12.157.3 Member Function Documentation .............................. 1842
12.157.3.1 cursor_color() [1/2] ....................................... 1842
12.157.3.2 cursor_color() [2/2] ....................................... 1842
12.157.3.3 draw() ........................................................... 1842
12.157.3.4 handle() ........................................................... 1842
12.157.3.5 resize() ........................................................... 1843
12.157.3.6 shortcut() [1/2] ................................................. 1843
12.157.3.7 shortcut() [2/2] ................................................. 1844
12.157.3.8 soft() ............................................................. 1844
12.157.3.9 textcolor() .......................................................... 1844
12.160.4.48 do_callback() [1/3] ......................................................... 1886
12.160.4.49 do_callback() [2/3] ......................................................... 1886
12.160.4.50 do_callback() [3/3] ......................................................... 1886
12.160.4.51 draw() .............................................................. 1887
12.160.4.52 draw_focus() [1/3] ......................................................... 1887
12.160.4.53 draw_focus() [2/3] ......................................................... 1887
12.160.4.54 draw_focus() [3/3] ......................................................... 1888
12.160.4.55 draw_label() [1/3] ......................................................... 1888
12.160.4.56 draw_label() [2/3] ......................................................... 1888
12.160.4.57 draw_label() [3/3] ......................................................... 1889
12.160.4.58 h() [1/2] .............................................................. 1889
12.160.4.59 h() [2/2] .............................................................. 1889
12.160.4.60 handle() .............................................................. 1889
12.160.4.61 hide() .............................................................. 1890
12.160.4.62 image() [1/4] .............................................................. 1890
12.160.4.63 image() [2/4] .............................................................. 1890
12.160.4.64 image() [3/4] .............................................................. 1890
12.160.4.65 image() [4/4] .............................................................. 1890
12.160.4.66 image_bound() ......................................................... 1891
12.160.4.67 inside() .............................................................. 1891
12.160.4.68 is_label_copied() ...................................................... 1891
12.160.4.69 label() [1/3] .............................................................. 1892
12.160.4.70 label() [2/3] .............................................................. 1892
12.160.4.71 label() [3/3] .............................................................. 1892
12.160.4.72 label_shortcut() ...................................................... 1892
12.160.4.73 labelcolor() [1/2] ......................................................... 1893
12.160.4.74 labelcolor() [2/2] ......................................................... 1893
12.160.4.75 labelfont() [1/2] ......................................................... 1893
12.160.4.76 labelfont() [2/2] ......................................................... 1893
12.160.4.77 labelsize() [1/2] ......................................................... 1894
12.160.4.78 labelsize() [2/2] ......................................................... 1894
12.160.4.79 labeltype() [1/2] ......................................................... 1894
12.160.4.80 labeltype() [2/2] ......................................................... 1894
12.160.4.81 measure_label() ...................................................... 1895
12.160.4.82 needs_keyboard() [1/2] .................................................. 1895
12.160.4.83 needs_keyboard() [2/2] .................................................. 1895
12.160.4.84 output() .............................................................. 1895
12.160.4.85 parent() [1/2] .............................................................. 1896
12.160.4.86 parent() [2/2] .............................................................. 1896
12.160.4.87 position() .............................................................. 1896
12.160.4.88 redraw() .............................................................. 1896
12.160.4.89 redraw_label() ......................................................... 1896
### 12.161.3.2 draw_decorated_window() ........................................................................ 1907
### 12.161.3.3 origin() [1/2] ....................................................................................... 1907
### 12.161.3.4 origin() [2/2] ....................................................................................... 1907
### 12.161.3.5 print_window_part() ............................................................................. 1908
### 12.161.3.6 printable_rect() ................................................................................... 1908
### 12.161.3.7 translate() ........................................................................................... 1908
### 12.161.3.8 untranslate() ....................................................................................... 1909

### 12.162 Fl_Widget_Tracker Class Reference ................................................................. 1909
### 12.162.1 Detailed Description ................................................................................... 1909
### 12.162.2 Member Function Documentation ............................................................. 1910

#### 12.162.2.1 deleted() ............................................................................................. 1910
#### 12.162.2.2 exists() ............................................................................................... 1910
#### 12.162.2.3 widget() .............................................................................................. 1910

### 12.163 Fl_Window Class Reference ............................................................................ 1910
### 12.163.1 Detailed Description ................................................................................... 1922
### 12.163.2 Constructor & Destructor Documentation .................................................. 1922

#### 12.163.2.1 Fl_Window() [1/2] ................................................................................. 1922
#### 12.163.2.2 Fl_Window() [2/2] ................................................................................. 1922
#### 12.163.2.3 ~Fi_Window() ......................................................................................... 1923

### 12.163.3 Member Function Documentation .............................................................. 1923

#### 12.163.3.1 as_double_window() ......................................................................... 1923
#### 12.163.3.2 as_overlay_window() .......................................................................... 1923
#### 12.163.3.3 as_window() [1/2] ................................................................................ 1923
#### 12.163.3.4 as_window() [2/2] ................................................................................ 1923
#### 12.163.3.5 border() .............................................................................................. 1924
#### 12.163.3.6 clear_border() ..................................................................................... 1924
#### 12.163.3.7 clear_modal_states() .......................................................................... 1924
#### 12.163.3.8 current() ............................................................................................. 1924
#### 12.163.3.9 cursor() [1/3] ....................................................................................... 1925
#### 12.163.3.10 cursor() [2/3] ..................................................................................... 1925
#### 12.163.3.11 cursor() [3/3] ..................................................................................... 1925
#### 12.163.3.12 decorated_h() .................................................................................... 1925
#### 12.163.3.13 decorated_w() .................................................................................... 1926
#### 12.163.3.14 default_cursor() [1/2] ..................................................................... 1926
#### 12.163.3.15 default_cursor() [2/2] ..................................................................... 1926
#### 12.163.3.16 default_icon() ................................................................................... 1926
#### 12.163.3.17 default_icons() [1/2] ..................................................................... 1926
#### 12.163.3.18 default_icons() [2/2] ..................................................................... 1927
#### 12.163.3.19 default_size_range() ....................................................................... 1927
#### 12.163.3.20 default_xclass() [1/2] ..................................................................... 1928
#### 12.163.3.21 default_xclass() [2/2] ..................................................................... 1928
#### 12.163.3.22 draw() .............................................................................................. 1929
flush() .................................................. 1929
force_position() [1/2] ................................. 1929
force_position() [2/2] ................................. 1929
free_icons() ............................................. 1929
free_position() ........................................ 1930
fullscreen() ........................................... 1930
fullscreen_screens() ................................. 1930
handle() ................................................. 1930
hide() .................................................... 1931
hotspot() ............................................... 1931
icon() [1/3] ............................................... 1931
icon() [2/3] ............................................... 1931
icon() [3/3] ............................................... 1932
iconize() ............................................... 1932
icons() [1/2] ............................................ 1932
icons() [2/2] ............................................ 1933
is_resizable() .......................................... 1933
make_current() ........................................ 1933
maximize() .............................................. 1933
modal() .................................................. 1934
os_id() .................................................. 1934
resize() ............................................... 1934
screen_num() .......................................... 1934
set_menu_window() .................................... 1935
set_modal() ............................................. 1935
set_non_modal() ....................................... 1935
set_tooltip_window() ................................. 1935
shape() [1/2] ............................................ 1935
shape() [2/2] ............................................ 1935
show() [1/2] ............................................. 1936
show() [2/2] ............................................. 1937
show_next_window_iconic() [1/2] ................... 1937
show_next_window_iconic() [2/2] ................... 1937
shown() .................................................. 1937
size_range() ............................................ 1937
un_maximize() ......................................... 1938
wait_for_expose() .................................... 1938
xclass() [1/2] .......................................... 1939
xclass() [2/2] .......................................... 1939

12.163.4 Member Data Documentation ............... 1940

12.164 Fl_Wizard Class Reference ................. 1940
12.183 Fl_Text_Display::Style_Table_Entry Struct Reference .................................. 1963
12.183.1 Detailed Description ......................................................................................... 1963
12.184 Fl_Terminal::Utf8Char Class Reference ............................................................ 1963

13 File Documentation .................................................................................................. 1965
13.1 Enumerations.H File Reference ................................................................................ 1965
  13.1.1 Detailed Description .............................................................................................. 1976
  13.1.2 Macro Definition Documentation ........................................................................ 1976
     13.1.2.1 FL_ABI_VERSION ......................................................................................... 1976
     13.1.2.2 FL_API_VERSION ......................................................................................... 1977
     13.1.2.3 FL_IMAGE_LABEL ....................................................................................... 1977
     13.1.2.4 FL_MAJOR_VERSION ...................................................................................... 1977
     13.1.2.5 FL_MINOR_VERSION ...................................................................................... 1977
     13.1.2.6 FL_MULTI_LABEL ......................................................................................... 1977
     13.1.2.7 FL_PATCH_VERSION ....................................................................................... 1977
     13.1.2.8 FL_SYMBOL_LABEL ....................................................................................... 1978
     13.1.2.9 FL_VERSION .................................................................................................. 1978
  13.1.3 Typedef Documentation ....................................................................................... 1978
     13.1.3.1 Fl_Contrast_Function ..................................................................................... 1978
     13.1.3.2 Fl_Fontsize ................................................................................................... 1978
  13.1.4 Enumeration Type Documentation ...................................................................... 1978
     13.1.4.1 anonymous enum .......................................................................................... 1978
     13.1.4.2 Fl_Arrow_Type ............................................................................................. 1979
     13.1.4.3 Fl_Boxtype ................................................................................................... 1979
     13.1.4.4 Fl_Callback_Reason ..................................................................................... 1981
     13.1.4.5 Fl_Contrast_Mode .......................................................................................... 1982
     13.1.4.6 Fl_Cursor ...................................................................................................... 1982
     13.1.4.7 Fl_Damage ................................................................................................... 1983
     13.1.4.8 Fl_Event ........................................................................................................ 1983
     13.1.4.9 Fl_Labeltype ................................................................................................. 1986
     13.1.4.10 Fl_Orientation ............................................................................................. 1986
     13.1.4.11 Fl_When ...................................................................................................... 1987
  13.1.5 Function Documentation .................................................................................... 1988
     13.1.5.1 fl_box() ....................................................................................................... 1988
     13.1.5.2 fl_color_cube() ........................................................................................... 1988
     13.1.5.3 fl_define_FL_EMBOSSED_LABEL() .............................................................. 1988
     13.1.5.4 fl_define_FL_ENGRAVED_LABEL() ............................................................. 1988
     13.1.5.5 fl_define_FL_ICON_LABEL() ...................................................................... 1988
     13.1.5.6 fl_define_FL_IMAGE_LABEL() ................................................................. 1988
     13.1.5.7 fl_define_FL_MULTI_LABEL() ................................................................. 1988
     13.1.5.8 fl_define_FL_SHADOW_LABEL() .............................................................. 1988
     13.1.5.9 fl_down() .................................................................................................... 1989
<table>
<thead>
<tr>
<th>Section</th>
<th>File Reference</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.24</td>
<td>fl_callback_macros.H</td>
<td>2025</td>
</tr>
<tr>
<td>13.24.1</td>
<td>Detailed Description</td>
<td>2026</td>
</tr>
<tr>
<td>13.24.2</td>
<td>Macro Definition Documentation</td>
<td>2026</td>
</tr>
<tr>
<td>13.24.2.1</td>
<td>FL_FUNCTION_CALLBACK_3</td>
<td>2026</td>
</tr>
<tr>
<td>13.24.2.2</td>
<td>FL_INLINE_CALLBACK_2</td>
<td>2026</td>
</tr>
<tr>
<td>13.24.2.3</td>
<td>FL_METHOD_CALLBACK_1</td>
<td>2027</td>
</tr>
<tr>
<td>13.25</td>
<td>fl_callback_macros.H</td>
<td>2028</td>
</tr>
<tr>
<td>13.26</td>
<td>fl_casts.H</td>
<td>2034</td>
</tr>
<tr>
<td>13.27</td>
<td>Fl_Chart.H File Reference</td>
<td>2034</td>
</tr>
<tr>
<td>13.27.1</td>
<td>Detailed Description</td>
<td>2035</td>
</tr>
<tr>
<td>13.28</td>
<td>Fl_Chart.H</td>
<td>2035</td>
</tr>
<tr>
<td>13.29</td>
<td>Fl_Check_Browser.H</td>
<td>2036</td>
</tr>
<tr>
<td>13.30</td>
<td>Fl_Check_Button.H</td>
<td>2037</td>
</tr>
<tr>
<td>13.31</td>
<td>FlChoice.H</td>
<td>2038</td>
</tr>
<tr>
<td>13.32</td>
<td>Fl_Clock.H</td>
<td>2038</td>
</tr>
<tr>
<td>13.33</td>
<td>Fl_Color_Chooser.H File Reference</td>
<td>2039</td>
</tr>
<tr>
<td>13.33.1</td>
<td>Detailed Description</td>
<td>2040</td>
</tr>
<tr>
<td>13.34</td>
<td>Fl_Color_Chooser.H</td>
<td>2040</td>
</tr>
<tr>
<td>13.35</td>
<td>fl_config.h</td>
<td>2041</td>
</tr>
<tr>
<td>13.36</td>
<td>Fl_Copy_Surface.H</td>
<td>2042</td>
</tr>
<tr>
<td>13.37</td>
<td>Fl.Counter.H</td>
<td>2043</td>
</tr>
<tr>
<td>13.38</td>
<td>Fl_Device.H File Reference</td>
<td>2044</td>
</tr>
<tr>
<td>13.38.1</td>
<td>Detailed Description</td>
<td>2044</td>
</tr>
<tr>
<td>13.39</td>
<td>Fl_Device.H</td>
<td>2044</td>
</tr>
<tr>
<td>13.40</td>
<td>Fl_Dial.H</td>
<td>2045</td>
</tr>
<tr>
<td>13.41</td>
<td>Fl_Double_Window.H</td>
<td>2046</td>
</tr>
<tr>
<td>13.42</td>
<td>fl_draw.H File Reference</td>
<td>2046</td>
</tr>
<tr>
<td>13.42.1</td>
<td>Detailed Description</td>
<td>2052</td>
</tr>
<tr>
<td>13.43</td>
<td>fl_draw.H</td>
<td>2052</td>
</tr>
<tr>
<td>13.44</td>
<td>Fl_Export.H</td>
<td>2058</td>
</tr>
<tr>
<td>13.45</td>
<td>Fl_File_Browser.H</td>
<td>2058</td>
</tr>
<tr>
<td>13.46</td>
<td>Fl_File_Chooser.H</td>
<td>2059</td>
</tr>
<tr>
<td>13.47</td>
<td>Fl_File_Icon.H</td>
<td>2061</td>
</tr>
<tr>
<td>13.48</td>
<td>Fl_File_Input.H</td>
<td>2063</td>
</tr>
<tr>
<td>13.49</td>
<td>Fl_Fill_Dial.H</td>
<td>2063</td>
</tr>
<tr>
<td>13.50</td>
<td>Fl_Fill_Slider.H</td>
<td>2064</td>
</tr>
<tr>
<td>13.51</td>
<td>Fl_Flex.H</td>
<td>2064</td>
</tr>
<tr>
<td>13.52</td>
<td>Fl_Float_Input.H</td>
<td>2066</td>
</tr>
<tr>
<td>13.53</td>
<td>Fl_FormsBitmap.H</td>
<td>2066</td>
</tr>
<tr>
<td>13.54</td>
<td>Fl_FormsPixmap.H</td>
<td>2067</td>
</tr>
<tr>
<td>13.55</td>
<td>Fl_Free.H</td>
<td>2067</td>
</tr>
<tr>
<td>13.56</td>
<td>Fl_GIF_Image.H</td>
<td>2068</td>
</tr>
<tr>
<td>Section</td>
<td>File Name</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>13.92</td>
<td>Fl_Multiline_Output.H</td>
<td>2106</td>
</tr>
<tr>
<td>13.93</td>
<td>Fl_Native_File_Chooser.H</td>
<td>2107</td>
</tr>
<tr>
<td></td>
<td>13.93.1 Detailed Description</td>
<td>2107</td>
</tr>
<tr>
<td>13.94</td>
<td>Fl_Native_File_Chooser.H</td>
<td>2107</td>
</tr>
<tr>
<td>13.95</td>
<td>Fl_Nice_Slider.H</td>
<td>2109</td>
</tr>
<tr>
<td>13.96</td>
<td>Fl_Object.H</td>
<td>2110</td>
</tr>
<tr>
<td>13.97</td>
<td>Fl_Output.H</td>
<td>2110</td>
</tr>
<tr>
<td>13.98</td>
<td>Fl_Overlay_Window.H</td>
<td>2110</td>
</tr>
<tr>
<td>13.99</td>
<td>Fl_Pack.H</td>
<td>2111</td>
</tr>
<tr>
<td>13.100</td>
<td>Fl_Paged_Device.H File Reference</td>
<td>2112</td>
</tr>
<tr>
<td></td>
<td>13.100.1 Detailed Description</td>
<td>2112</td>
</tr>
<tr>
<td>13.101</td>
<td>Fl_Paged_Device.H</td>
<td>2112</td>
</tr>
<tr>
<td>13.102</td>
<td>Fl_Pixmap.H</td>
<td>2113</td>
</tr>
<tr>
<td>13.103</td>
<td>Fl_Plugin.H</td>
<td>2114</td>
</tr>
<tr>
<td>13.104</td>
<td>Fl_PNG_Imagge.H</td>
<td>2115</td>
</tr>
<tr>
<td>13.105</td>
<td>Fl_PNM_Imagge.H</td>
<td>2115</td>
</tr>
<tr>
<td>13.106</td>
<td>Fl_Positioner.H</td>
<td>2116</td>
</tr>
<tr>
<td>13.107</td>
<td>Fl_PostScript.H File Reference</td>
<td>2117</td>
</tr>
<tr>
<td></td>
<td>13.107.1 Detailed Description</td>
<td>2117</td>
</tr>
<tr>
<td></td>
<td>13.107.2 Typedef Documentation</td>
<td>2117</td>
</tr>
<tr>
<td></td>
<td>13.107.2.1 Fl_PostScript_Close_Command</td>
<td>2117</td>
</tr>
<tr>
<td>13.108</td>
<td>Fl_PostScript.H</td>
<td>2117</td>
</tr>
<tr>
<td>13.109</td>
<td>Fl_Preferences.H</td>
<td>2118</td>
</tr>
<tr>
<td>13.110</td>
<td>Fl_Printer.H File Reference</td>
<td>2121</td>
</tr>
<tr>
<td></td>
<td>13.110.1 Detailed Description</td>
<td>2122</td>
</tr>
<tr>
<td>13.111</td>
<td>Fl_Printer.H</td>
<td>2122</td>
</tr>
<tr>
<td>13.112</td>
<td>Fl_Progress.H</td>
<td>2123</td>
</tr>
<tr>
<td>13.113</td>
<td>Fl_Radio_Button.H</td>
<td>2123</td>
</tr>
<tr>
<td>13.114</td>
<td>Fl_Radio_Light_Button.H</td>
<td>2124</td>
</tr>
<tr>
<td>13.115</td>
<td>Fl_Radio_Round_Button.H</td>
<td>2124</td>
</tr>
<tr>
<td>13.116</td>
<td>Fl_Rect.H</td>
<td>2125</td>
</tr>
<tr>
<td>13.117</td>
<td>Fl_Repeat_Button.H</td>
<td>2126</td>
</tr>
<tr>
<td>13.118</td>
<td>Fl_Return_Button.H</td>
<td>2126</td>
</tr>
<tr>
<td>13.119</td>
<td>Fl_RGB_Imagge.H</td>
<td>2127</td>
</tr>
<tr>
<td>13.120</td>
<td>Fl_Roller.H</td>
<td>2127</td>
</tr>
<tr>
<td>13.121</td>
<td>Fl_Round_Button.H</td>
<td>2127</td>
</tr>
<tr>
<td>13.122</td>
<td>Fl_Round_Clock.H</td>
<td>2128</td>
</tr>
<tr>
<td>13.123</td>
<td>Fl_Scheme.H</td>
<td>2128</td>
</tr>
<tr>
<td>13.124</td>
<td>Fl_Scheme_Choice.H</td>
<td>2129</td>
</tr>
<tr>
<td>13.125</td>
<td>Fl_Scroll.H</td>
<td>2129</td>
</tr>
<tr>
<td>13.126</td>
<td>Fl_Scrollbar.H</td>
<td>2131</td>
</tr>
<tr>
<td>13.127</td>
<td>Fl_Secret_Input.H</td>
<td>2131</td>
</tr>
</tbody>
</table>

Generated by Doxygen
13.160.1 Detailed Description .................................................. 2176
13.160.2 Enumeration Type Documentation .................................. 2176
  13.160.2.1 Fl_Tree_Reason .................................................. 2176
13.161 Fl_Tree.H ................................................................. 2176
13.162 Fl_Tree_Item.H File Reference ...................................... 2179
  13.162.1 Detailed Description ................................................ 2179
13.163 Fl_Tree_Item.H .......................................................... 2179
13.164 Fl_Tree_Item_Array.H File Reference ............................... 2183
  13.164.1 Detailed Description ................................................ 2183
13.165 Fl_Tree_Item_Array.H ................................................... 2184
13.166 Fl_Tree_Prefs.H File Reference ...................................... 2184
  13.166.1 Detailed Description ................................................ 2185
  13.166.2 Enumeration Type Documentation .................................. 2185
    13.166.2.1 Fl_Tree_Connector ............................................. 2185
    13.166.2.2 Fl_Tree_Item_Draw_Mode ..................................... 2185
    13.166.2.3 Fl_Tree_Item_Reselect_Mode ................................ 2186
    13.166.2.4 Fl_Tree_Select ............................................... 2186
    13.166.2.5 Fl_Tree_Sort .................................................. 2186
13.167 Fl_Tree_Prefs.H ...................................................... 2186
13.168 fl_types.h File Reference ......................................... 2190
  13.168.1 Detailed Description ................................................ 2190
  13.168.2 Typedef Documentation .......................................... 2190
    13.168.2.1 Fl_Shortcut .................................................. 2190
13.169 fl_types.h ............................................................. 2190
13.170 fl_utf8.h File Reference ........................................... 2191
  13.170.1 Detailed Description ................................................ 2193
13.171 fl_utf8.h ............................................................... 2193
13.172 Fl_Valuator.H .......................................................... 2195
13.173 Fl_Value_Input.H ....................................................... 2196
13.174 Fl_Value_Output.H ...................................................... 2197
13.175 Fl_Value_Slider.H ....................................................... 2198
13.176 Fl_Widget.H File Reference ........................................ 2199
  13.176.1 Detailed Description ................................................ 2199
  13.176.2 Macro Definition Documentation .................................. 2199
    13.176.2.1 FL_RESERVED_TYPE ......................................... 2199
13.177 Fl_Widget.H ............................................................. 2199
13.178 Fl_Widget_Surface.H .................................................... 2204
13.179 Fl_Window.H File Reference ........................................ 2205
  13.179.1 Detailed Description ................................................ 2205
13.180 Fl_Window.H ............................................................. 2205
13.181 Fl_Wizard.H ............................................................. 2208
13.182 Fl_XBM_Image.H ........................................................ 2208
<table>
<thead>
<tr>
<th>Section</th>
<th>File</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.241</td>
<td>FL_XColor.H</td>
<td>2283</td>
</tr>
<tr>
<td>13.242</td>
<td>fstring.h</td>
<td>2283</td>
</tr>
<tr>
<td>13.243</td>
<td>freeglut_teatop_data.h</td>
<td>2284</td>
</tr>
<tr>
<td>13.244</td>
<td>gtk-shell-client-protocol.h</td>
<td>2286</td>
</tr>
<tr>
<td>13.245</td>
<td>mediumarrow.h</td>
<td>2289</td>
</tr>
<tr>
<td>13.246</td>
<td>numericsort.c</td>
<td>2289</td>
</tr>
<tr>
<td></td>
<td>Function Documentation</td>
<td>2290</td>
</tr>
<tr>
<td></td>
<td>fl_casenumericsort()</td>
<td>2290</td>
</tr>
<tr>
<td></td>
<td>fl_numericsort()</td>
<td>2290</td>
</tr>
<tr>
<td>13.247</td>
<td>print_button.h</td>
<td>2291</td>
</tr>
<tr>
<td>13.248</td>
<td>print_panel.h</td>
<td>2291</td>
</tr>
<tr>
<td>13.249</td>
<td>slowarrow.h</td>
<td>2292</td>
</tr>
<tr>
<td>13.250</td>
<td>utf8_internal.h</td>
<td>2292</td>
</tr>
<tr>
<td>13.251</td>
<td>vsnprintf.c</td>
<td>2293</td>
</tr>
<tr>
<td></td>
<td>Function Documentation</td>
<td>2293</td>
</tr>
<tr>
<td></td>
<td>fl_vsnprintf()</td>
<td>2293</td>
</tr>
<tr>
<td>13.252</td>
<td>Xutf8.h</td>
<td>2294</td>
</tr>
<tr>
<td>13.253</td>
<td>case.h</td>
<td>2296</td>
</tr>
<tr>
<td>13.254</td>
<td>dingbats_.h</td>
<td>2316</td>
</tr>
<tr>
<td>13.255</td>
<td>spacing.h</td>
<td>2323</td>
</tr>
<tr>
<td>13.256</td>
<td>symbol_.h</td>
<td>2345</td>
</tr>
<tr>
<td>13.257</td>
<td>imKStoUCS.c</td>
<td>2358</td>
</tr>
<tr>
<td>13.258</td>
<td>armsgci_8.h</td>
<td>2362</td>
</tr>
<tr>
<td>13.259</td>
<td>ascii.h</td>
<td>2363</td>
</tr>
<tr>
<td>13.260</td>
<td>big5.h</td>
<td>2364</td>
</tr>
<tr>
<td>13.261</td>
<td>big5_emacs.h</td>
<td>2411</td>
</tr>
<tr>
<td>13.262</td>
<td>cp1133.h</td>
<td>2413</td>
</tr>
<tr>
<td>13.263</td>
<td>cp1251.h</td>
<td>2414</td>
</tr>
<tr>
<td>13.264</td>
<td>cp1255.h</td>
<td>2416</td>
</tr>
<tr>
<td>13.265</td>
<td>cp1256.h</td>
<td>2417</td>
</tr>
<tr>
<td>13.266</td>
<td>cp936ext.h</td>
<td>2419</td>
</tr>
<tr>
<td>13.267</td>
<td>gb2312.h</td>
<td>2491</td>
</tr>
<tr>
<td>13.268</td>
<td>georgian_academy.h</td>
<td>2520</td>
</tr>
<tr>
<td>13.269</td>
<td>georgian_ps.h</td>
<td>2521</td>
</tr>
<tr>
<td>13.270</td>
<td>iso8859_1.h</td>
<td>2523</td>
</tr>
<tr>
<td>13.271</td>
<td>iso8859_10.h</td>
<td>2523</td>
</tr>
<tr>
<td>13.272</td>
<td>iso8859_11.h</td>
<td>2524</td>
</tr>
<tr>
<td>13.273</td>
<td>iso8859_13.h</td>
<td>2525</td>
</tr>
<tr>
<td>13.274</td>
<td>iso8859_14.h</td>
<td>2526</td>
</tr>
<tr>
<td>13.275</td>
<td>iso8859_15.h</td>
<td>2528</td>
</tr>
<tr>
<td>13.276</td>
<td>iso8859_16.h</td>
<td>2528</td>
</tr>
</tbody>
</table>
13.277 iso8859_2.h ................................................................. 2530
13.278 iso8859_3.h ................................................................. 2531
13.279 iso8859_4.h ................................................................. 2532
13.280 iso8859_5.h ................................................................. 2533
13.281 iso8859_6.h ................................................................. 2534
13.282 iso8859_7.h ................................................................. 2535
13.283 iso8859_8.h ................................................................. 2536
13.284 iso8859_9.h ................................................................. 2537
13.285 iso8859_9e.h ............................................................... 2538
13.286 jisx0201.h ................................................................. 2540
13.287 jisx0208.h ................................................................. 2540
13.288 jisx0212.h ................................................................. 2540
13.289 koI8_c.h ................................................................. 2593
13.290 koI8_r.h ................................................................. 2594
13.291 koI8_u.h ................................................................. 2596
13.292 ksc5601.h ............................................................... 2598
13.293 mulelao.h ............................................................... 2632
13.294 tatar_cyr.h ............................................................... 2633
13.295 tcvn.h ................................................................. 2635
13.296 tis620.h ................................................................. 2636
13.297 ucs2be.h ................................................................. 2637
13.298 utf8.h ................................................................. 2638
13.299 viscii.h ................................................................. 2639
13.300 mk_wcwidth.c ........................................................... 2640
13.301 ucs2fontmap.c .......................................................... 2645
13.302 utf8Utils.c ............................................................ 2649
13.303 Ximint.h ................................................................. 2651
13.304 Xlibint.h ................................................................. 2651

Index ................................................................. 2653
Chapter 1

FLTK Programming Manual

FLTK 1.4.0 Programming Manual
By F. Costantini, D. Gibson, M. Melcher, A. Schlosser, B. Spitzak and M. Sweet.
Copyright © 1998 - 2024 by Bill Spitzak and others.

This software and manual are provided under the terms of the GNU Library General Public License. Permission is granted to reproduce this manual or any portion for any purpose, provided this copyright and permission notice are preserved.
1.1 Preface

This manual describes the Fast Light Tool Kit ("FLTK") version 1.4.0, a C++ Graphical User Interface ("GUI") toolkit for UNIX, Microsoft Windows and Apple OS X.

Version 1.4.0 introduces support for a new windowing system under Linux/Unix: Wayland. FLTK applications under Linux/Unix run, unchanged, as Wayland or X11 clients depending on what’s available at run-time.

Each of the chapters in this manual is designed as a tutorial for using FLTK, while the appendices provide a convenient reference for all FLTK widgets, functions, and operating system interfaces.

This manual may be printed, modified, and/or used under the terms of the FLTK license provided in Software License.

1.1.1 Organization

This manual is organized into the following chapters and appendices:

- Introduction to FLTK
- FLTK Basics
- Common Widgets and Attributes
- Adding and Extending Widgets
- Using OpenGL
- Programming with FLUID
  - GUI Attributes
  - Selecting and Moving Widgets
  - Image Labels
- FLTK Runtime Options
- Advanced FLTK
- Unicode and UTF-8 Support
- Appendices:
  - Constants and Enumerations
  - GLUT Compatibility
    - class Fl_Glut_Window
  - Forms Compatibility
  - Operating System Issues
  - Migrating Code from FLTK 1.3 to 1.4
  - Software License
  - Example Source Code
  - FAQ (Frequently Asked Questions)
1.1 Preface

- Designing a Simple Text Editor
- Drawing Things in FLTK
- Handling Events
- Adding and Extending Widgets
- Using OpenGL
- Programming with FLUID
- FLTK Runtime Options
- Advanced FLTK
- Unicode and UTF-8 Support
- Constants and Enumerations
- GLUT Compatibility
- Forms Compatibility
- Operating System Issues
- Migrating Code from FLTK 1.3 to 1.4
- Developer Information
- Software License
- Example Source Code

1.1.2 Conventions

This manual was generated using Doxygen (see http://www.doxygen.org/) to process the source code itself, special comments in the code, and additional documentation files. In general, Doxygen recognizes and denotes the following entities as shown:

- classes, such as Fl_Widget,
- methods, such as Fl_Widget::callback(Fl_Callback *cb, void *p),
- functions, such as fl_draw(const char *str, int x, int y),
- internal links, such as Conventions,
- external links, such as http://www.stack.nl/~dimitri/doxygen/

Other code samples and commands are shown in regular courier type.
1.1.3 Abbreviations

The following abbreviations are used in this manual:

X11

The X Window System version 11.

Xlib

The X Window System interface library.

MS Windows, WIN32

The Microsoft Windows Application Programmer’s Interface for Windows 2000, Windows XP, Windows Vista, Windows 7 and later Windows versions. FLTK uses the preprocessor definition _WIN32 for the 32 bit and 64 bit MS Windows API.

OS X, APPLE

The Apple desktop operating system OS X 10.0 and later. MacOS 8 and 9 support was dropped after FLTK 1.0.10. FLTK uses the preprocessor definition APPLE for OS X.

1.1.4 Copyrights and Trademarks

FLTK is Copyright © 1998 - 2024 by Bill Spitzak and others. Use and distribution of FLTK is governed by the GNU Library General Public License with 4 exceptions, located in Software License.

UNIX is a registered trademark of the X Open Group, Inc. Microsoft and Windows are registered trademarks of Microsoft Corporation. OpenGL is a registered trademark of Silicon Graphics, Inc. Apple, Macintosh, MacOS, and Mac OS X are registered trademarks of Apple Computer, Inc.

1.2 Introduction to FLTK

The Fast Light Tool Kit ("FLTK") is a cross-platform C++ GUI toolkit for UNIX®/Linux® (X11 or Wayland), Microsoft® Windows®, and Apple® macOS®.

FLTK provides modern GUI functionality without the bloat and supports 3D graphics via OpenGL® and its built-in GLUT emulation. It was originally developed by Mr. Bill Spitzak and is currently maintained by a small group of developers across the world with a central repository in the US.
1.2 Introduction to FLTK

1.2.1 History of FLTK

It has always been Bill's belief that the GUI API of all modern systems is much too high level. Toolkits (even FLTK) are not what should be provided and documented as part of an operating system. The system only has to provide arbitrary shaped but featureless windows, a powerful set of graphics drawing calls, and a simple unalterable method of delivering events to the owners of the windows. NeXT (if you ignored NextStep) provided this, but they chose to hide it and tried to push their own baroque toolkit instead.

Many of the ideas in FLTK were developed on a NeXT (but not using NextStep) in 1987 in a C toolkit Bill called "views". Here he came up with passing events downward in the tree and having the handle routine return a value indicating whether it used the event, and the table-driven menus. In general he was trying to prove that complex UI ideas could be entirely implemented in a user space toolkit, with no knowledge or support by the system.

After going to film school for a few years, Bill worked at Sun Microsystems on the (doomed) NeWS project. Here he found an even better and cleaner windowing system, and he reimplemented "views" atop that. NeWS did have an unnecessarily complex method of delivering events which hurt it. But the designers did admit that perhaps the user could write just as good of a button as they could, and officially exposed the lower level interface.

With the death of NeWS Bill realized that he would have to live with X. The biggest problem with X is the "window manager", which means that the toolkit can no longer control the window borders or drag the window around.

At Digital Domain Bill discovered another toolkit, "Forms". Forms was similar to his work, but provided many more widgets, since it was used in many real applications, rather than as theoretical work. He decided to use Forms, except he integrated his table-driven menus into it. Several very large programs were created using this version of Forms.

The need to switch to OpenGL and GLX, portability, and a desire to use C++ subclassing required a rewrite of Forms. This produced the first version of FLTK. The conversion to C++ required so many changes it made it impossible to recompile any Forms objects. Since it was incompatible anyway, Bill decided to incorporate his older ideas as much as possible by simplifying the lower level interface and the event passing mechanism.

Bill received permission to release it for free on the Internet, with the GNU general public license. Response from Internet users indicated that the Linux market dwarfed the SGI and high-speed GL market, so he rewrote it to use X for all drawing, greatly speeding it up on these machines. That is the version you have now.

Digital Domain has since withdrawn support for FLTK. While Bill is no longer able to actively develop it, he still contributes to FLTK in his free time and is a part of the FLTK development team.

1.2.2 Features

FLTK was designed to be statically linked. This was done by splitting it into many small objects and designing it so that functions that are not used do not have pointers to them in the parts that are used, and thus do not get linked in. This allows you to make an easy-to-install program or to modify FLTK to the exact requirements of your application without worrying about bloat. FLTK works fine as a shared library, though, and is now included with several Linux distributions.

Here are some of the core features unique to FLTK:

- `sizeof(Fl_Widget) == 64 to 92.`

- The "core" (the "hello" program compiled & linked with a static FLTK library using gcc on a 486 and then stripped) is 114K.

- The FLUID program (which includes every widget) is 538K.
• Written directly atop core libraries (Xlib, Wayland, Windows or Cocoa) for maximum speed, and carefully optimized for code size and performance.

• Precise low-level compatibility between the X11, Windows and MacOS versions - only about 10% of the code is different.

• Interactive user interface builder program. Output is human-readable and editable C++ source code.

• Support for overlay hardware, with emulation if none is available.

• Very small & fast portable 2-D drawing library to hide Xlib, Cairo, Windows, or macOS Quartz.

• OpenGL/Mesa drawing area widget.

• Support for OpenGL overlay hardware on both X11 and Windows, with emulation if none is available.

• Text widgets with cut & paste, undo, and support for Unicode text and international input methods.

• Compatibility header file for the GLUT library.

• Compatibility header file for the XForms library.

1.2.3 Licensing

FLTK comes with complete free source code. FLTK is available under the terms of the GNU Library General Public License with exceptions that allow for static linking. Contrary to popular belief, it can be used in commercial software - even Bill Gates could use it!

1.2.4 What Does "FLTK" Mean?

FLTK was originally designed to be compatible with the Forms Library written for SGI machines. In that library all the functions and structures started with "fl_". This naming was extended to all new methods and widgets in the C++ library, and this prefix was taken as the name of the library. It is almost impossible to search for "FL" on the Internet, due to the fact that it is also the abbreviation for Florida. After much debating and searching for a new name for the toolkit, which was already in use by several people, Bill came up with "FLTK", including a bogus excuse that it stands for "The Fast Light Toolkit".

1.2.5 Building and Installing FLTK with CMake

Starting with version 1.4, the recommended FLTK building system is CMake. See file README.CMake of the FLTK source tree for all information. It's also possible to use configure and make as follows to build and install FLTK,
1.2.6 Building and Installing FLTK Under UNIX and macOS with make

In most cases you can just type "make". This will run configure with the default of no options and then compile everything.

FLTK uses GNU autoconf to configure itself for your UNIX platform. The main things that the configure script will look for are the X11 and OpenGL (or Mesa) header and library files. If these cannot be found in the standard include/library locations you’ll need to define the CFLAGS, CXXFLAGS, and LDFLAGS environment variables. For the Bourne and Korn shells you’d use:

```
CFLAGS=-Iincludedir; export CFLAGS
CXXFLAGS=-Iincludedir; export CXXFLAGS
LDFLAGS=-Llibdir; export LDFLAGS
```

For C shell and tcsh, use:

```
setenv CFLAGS "-Iincludedir"
setenv CXXFLAGS "-Iincludedir"
setenv LDFLAGS "-Llibdir"
```

By default configure will look for a C++ compiler named CC, c++, g++, or gcc in that order. To use another compiler you need to set the CXX environment variable:

```
CXX=xlC; export CXX
setenv CXX "xlC"
```

The CC environment variable can also be used to override the default C compiler (cc or gcc), which is used for a few FLTK source files.

You can run configure yourself to get the exact setup you need. Type ".configure <options>". where options are:

- **--enable-cygwin**
  
  Enable the Cygwin libraries under Windows

- **--enable-debug**

  Enable debugging code & symbols

- **--disable-gl**

  Disable OpenGL support

- **--disable-svg**

  Disable support of reading and writing of Support Vector Graphics (.svg) files.

- **--disable-print**

  Disable print support for an X11/Wayland platform

- **--enable-shared**

  Enable generation of shared libraries
–enable-threads

Enable multithreading support

–enable-wayland

This is the default for Linux and FreeBSD systems equipped with the Wayland software. Enable the use of Wayland for all window operations, of Cairo for all graphics and of Pango for text drawing. Resulting FLTK apps run as Wayland clients if a Wayland compositor is available at run-time, and as X11 clients otherwise but keep using Cairo and Pango for all graphics.

–disable-xft

Disables the Xft library, resulting in non anti-aliased fonts (X11 platform).

–enable-usecairo

All drawing operations use the Cairo library (rather than Xlib) producing antialiased graphics (X11 platform, implies –enable-pango).

–enable-pango

Enable the Pango library for drawing any text in any script with any font under X11/Wayland.

–enable-x11

When targeting cygwin, build with X11 GUI instead of windows GDI. Also applicable to macOS platforms supplemented with XQuartz.

–enable-cairo

Enable support of class Fl_Cairo_Window (all platforms, requires the Cairo library).

–enable-cairoext

Enable the FLTK instrumentation for cairo extended use (requires –enable-cairo).

–disable-gdiplus

Don't use GDI+ when drawing curves and oblique lines (Windows platform).

–enable-cp936

Under X11, enable use of the GB2312 locale

–bindir=/path

Set the location for executables [default = $prefix/bin]
1.2 Introduction to FLTK

--datadir=/path
Set the location for data files. [default = $prefix/share]

--libdir=/path
Set the location for libraries [default = $prefix/lib]

--includedir=/path
Set the location for include files. [default = $prefix/include]

--mandir=/path
Set the location for man pages. [default = $prefix/man]

--prefix=/dir
Set the directory prefix for files [default = /usr/local]

When the configure script is done you can just run the "make" command. This will build the library, FLUID tool, and all of the test programs.

To install the library, become root and type "make install". This will copy the "fluid" executable to "bindir", the header files to "includedir", and the library files to "libdir".

1.2.7 Building FLTK Under Microsoft Windows

NOTE: This documentation section is currently under review. More up-to-date information for this release may be available in the file "README.Windows.txt" and you should read that file to determine if there are changes that may be applicable to your build environment.

FLTK 1.3 is officially supported on Windows (2000,) 2003, XP, and later. Older Windows versions prior to Windows 2000 are not officially supported, but may still work. The main reason is that the OS version needs to support UTF-8. FLTK 1.3 is known to work on recent versions of Windows such as Windows 7, Windows 8/8.1 and Windows 10 and has been reported to work in both 32-bit and 64-bit versions of these.

FLTK currently supports the following development environments on the Windows platform:

CAUTION: Libraries built by any one of these build environments can not be mixed with object files from any of the other environments! (They use incompatible C++ conventions internally.)

Free Microsoft Visual C++ 2008 Express and Visual C++ 2010 Express or later versions using workspace and project files generated by CMake. Older versions and the commercial versions can be used as well, if they can open the project files. Be sure to get your service packs!

Since FLTK 1.4 the project files MUST be generated with CMake. Please read "README.CMake.txt" for more information about this.
1.2.7.1 GNU toolsets (Cygwin or MinGW) hosted on Windows

If using Cygwin with the Cygwin shell, or MinGW with the Msys shell, these build environments behave very much like a Unix or macOS build and the notes above in the section on Building and Installing FLTK Under UNIX and Apple macOS apply, in particular the descriptions of using the "configure" script and its related options.

In general for a build using these tools, e.g. for the Msys shell with MinGW, it should suffice to "cd" into the directory where you have extracted the FLTK tarball and type:

```
./configure
make
```

This will build the FLTK libraries and they can then be utilised directly from the build location. NOTE: this may be simpler than "installing" them in many cases as different tool chains on Windows have different ideas about where the files should be "installed" to.

For example, if you "install" the libraries using Msys/MinGW with the following command:

```
makemake install
```

Then Msys will "install" the libraries to where it thinks the path "/usr/local/" leads to. If you only ever build code from within the Msys environment this works well, but the actual "Windows path" these files are located in will be something like "C:\msys\1.0\local\lib", depending on where your Msys installation is rooted, which may not be useful to other tools.

If you want to install your built FLTK libraries in a non-standard location you may do:

```
sh configure --prefix=C:/FLTK
make
```

Where the value passed to "prefix" is the path at which you would like FLTK to be installed.

A subsequent invocation of "make install" will then place the FLTK libraries and header files into that path.

The other options to "configure" may also be used to tailor the build to suit your environment.

1.2.7.2 Using the Visual C++ DLL Library

The "fltkdll.dsp" project file builds a DLL-version of the FLTK library. Because of name mangling differences between PC compilers (even between different versions of Visual C++) you can only use the DLL that is generated with the same version compiler that you built it with.

When compiling an application or DLL that uses the FLTK DLL, you will need to define the FL_DLL preprocessor symbol to get the correct linkage commands embedded within the FLTK header files.

1.2.8 Internet Resources

FLTK is available on the 'net in a bunch of locations:

WWW

https://www.fltk.org/
https://www.fltk.org/bugs.php [for reporting bugs]
https://www.fltk.org/software.php [download source code]
https://www.fltk.org/newsgroups.php [newsgroup/forums]

NNTP Newsgroups

https://groups.google.com/forum/#!forum/fltkgeneral [Google Groups interface]
news://fltk.org:1024/ [NNTP interface]
https://www.fltk.org/newsgroups.php [web interface]
1.2.9 Reporting Bugs

To report a bug in FLTK, or for feature requests, please use the form at https://www.fltk.org/bugs.php, and click on "Submit Bug or Feature Request".

You'll be prompted for the FLTK version, operating system & version, and compiler that you are using. We will be unable to provide any kind of help without that basic information.

For general support and questions, please use the fltk.general newsgroup (see above, "NNTP Newsgroups") or the web interface to the newsgroups at https://www.fltk.org/newsgroups.php.

1.3 FLTK Basics

This chapter teaches you the basics of writing and compiling programs that use FLTK.

1.3.1 Writing Your First FLTK Program

Up to FLTK 1.3.x all FLTK programs were required to include the file <FL/Fl.H> as the first FLTK header file.

Since FLTK 1.4.0 this requirement was relaxed and <FL/Fl.H> needs only be included if the class Fl is used or if some other stuff like enumerations is used in the source code. Example code in this documentation may still include it "everywhere" even if it is no longer strictly required.

In addition the program must include a header file for each FLTK class it uses. Listing 1 shows a simple "Hello, World!" program that uses FLTK to display the window.

Listing 1 - "hello.cxx"

```c
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>

int main(int argc, char **argv) {
  Fl_Window *window = new Fl_Window(340, 180);
  Fl_Box *box = new Fl_Box(20, 40, 300, 100, "Hello, World!");
  box->box(FL_UP_BOX);
  box->labelfont(FL_BOLD + FL_ITALIC);
  box->labelsize(36);
  box->labeltype(FL_SHADOW_LABEL);
  window->end();
  window->show(argc, argv);
  return Fl::run();
}
```

After including the required header files, the program then creates a window. All following widgets will automatically be children of this window.

```c
Fl_Window *window = new Fl_Window(340, 180);
```

Then we create a box with the "Hello, World!" string in it. FLTK automatically adds the new box to window, the current grouping widget.

```c
Fl_Box *box = new Fl_Box(20, 40, 300, 100, "Hello, World!");
```

Next, we set the type of box and the font, size, and style of the label:

```c
box->box(FL_UP_BOX);
box->labelfont(FL_BOLD + FL_ITALIC);
box->labelsize(36);
box->labeltype(FL_SHADOW_LABEL);
```

We tell FLTK that we will not add any more widgets to window.

```c
window->end();
```
Finally, we show the window and enter the FLTK event loop:

```c
window->show(argc, argv);
return Fl::run();
```

The resulting program will display the "Hello, World!" window:

![Hello, World! Window](image)

Figure 1.1 The Hello, World! Window

You can quit the program by closing the window or pressing the **ESC**ape key.

### 1.3.1.1 Creating the Widgets

The widgets are created using the C++ `new` operator. For most widgets the arguments to the constructor are:

```
Fl_Widget(x, y, width, height, label)
```

The `x` and `y` parameters determine where the widget or window is placed on the screen. In FLTK the top left corner of the window or screen is the origin (i.e. `x = 0, y = 0`).

The `width` and `height` parameters determine the size of the widget or window. The maximum widget size is typically governed by the underlying window system or hardware.

**What Units Do FLTK Functions Use?** describes the unit FLTK employs for `x`, `y`, `width`, and `height`, and more generally, for all graphical quantities.

`label` is a pointer to a character string to label the widget with or `NULL`. If not specified the label defaults to `NULL`. The label string must be in static storage such as a string constant because FLTK does not make a copy of it - it just uses the pointer.

### 1.3.1.2 Creating Widget Hierarchies

Widgets are commonly ordered into functional groups, which in turn may be grouped again, creating a hierarchy of widgets. FLTK makes it easy to fill groups by automatically adding all widgets that are created between a `myGroup->begin()` and `myGroup->end()`. In this example, `myGroup` would be the current group.

Newly created groups and their derived widgets implicitly call `begin()` in the constructor, effectively adding all subsequently created widgets to itself until `end()` is called.

Calling `end()` on one group widget transfers the "current group" property to the parent of that widget. Calling `end()` on a top level window (which has no parent) sets the current group to `NULL`.

Setting the current group to `NULL` will stop automatic hierarchies. New widgets can now be added manually using `Fl_Group::add(...)` and `Fl_Group::insert(...).`
1.3 FLTK Basics

1.3.1.3 Get/Set Methods

box->box(FL_UP_BOX) sets the type of box the Fl_Box draws, changing it from the default of FL_NO_BOX, which means that no box is drawn. In our "Hello, World!" example we use FL_UP_BOX, which means that a raised button border will be drawn around the widget. More details are available in the Box Types section.

You could examine the boxtype by doing box->box(). FLTK uses method name overloading to make short names for get/set methods. A "set" method is always of the form "void name(type)", and a "get" method is always of the form "type name() const".

1.3.1.4 Redrawing After Changing Attributes

Almost all of the get/set pairs are very fast, short inline functions and thus very efficient. However, the "set" methods do not call redraw() - you have to call it yourself. This greatly reduces code size and execution time. The only common exceptions are value() which calls redraw() and label() which calls redraw_label() if necessary.

1.3.1.5 Labels

All widgets support labels. In the case of window widgets, the label is used for the label in the title bar. Our example program calls the labelfont(), labelsize(), and labeltype() methods.

The labelfont() method sets the typeface and style that is used for the label, which for this example we are using FL_BOLD and FL_ITALIC.

The labelsize() method sets the height of the font in FLTK units.

The labeltype() method sets the type of label. FLTK supports normal, embossed, and shadowed labels internally, and more types can be added as desired.

A complete list of all label options can be found in the section on Labels and Label Types.

1.3.1.6 Showing the Window

The show() method shows the widget or window. For windows you can also provide the command-line arguments to allow users to customize the appearance, size, and position of your windows.

1.3.1.7 The Main Event Loop

All FLTK applications (and most GUI applications in general) are based on a simple event processing model. User actions such as mouse movement, button clicks, and keyboard activity generate events that are sent to an application. The application may then ignore the events or respond to the user, typically by redrawing a button in the "down" position, adding the text to an input field, and so forth.

FLTK also supports idle, timer, and file pseudo-events that cause a function to be called when they occur. Idle functions are called when no user input is present and no timers or files need to be handled - in short, when the application is not doing anything. Idle callbacks are often used to update a 3D display or do other background processing.

Timer functions are called after a specific amount of time has expired. They can be used to pop up a progress dialog after a certain amount of time or do other things that need to happen at more-or-less regular intervals. FLTK timers are not 100% accurate, so they should not be used to measure time intervals, for example.

File functions are called when data is ready to read or write, or when an error condition occurs on a file. They are most often used to monitor network connections (sockets) for data-driven displays.

FLTK applications must periodically check (Fl::check()) or wait (Fl::wait()) for events or use the Fl::run() method to enter a standard event processing loop. Calling Fl::run() is equivalent to the following code:

```c
while (Fl::wait());
```

Fl::run() does not return until all of the windows under FLTK control are closed by the user or your program.
1.3.2 Naming Conventions

All public symbols in FLTK start with the characters 'F' and 'L':

- Functions are either `Fl::foo()` or `fl_foo()`.

- Class and type names are capitalized: `Fl_Foo`.

- Constants and Enumerations are uppercase: `FL_FOO`.

- All header files start with `<FL/...>`.

1.3.3 Header Files

The proper way to include FLTK header files is:

```
#include <FL/Fl_xyz.H>
```

**Note**

Case is significant on many operating systems, and the C standard uses the forward slash (/) to separate directories. Do not use any of the following include lines:

```
#include <FL/Fl_xyz.H>
#include <fl/fl_xyz.h>
#include <Fl/fl_xyz.h>
```

1.3.4 Compiling Programs that Use FLTK

Since FLTK 1.4 CMake is the recommended build system. The details below show the "old" methods and reference information in case you like to write your build configuration manually (e.g. Makefiles, Visual Studio, other IDE's ...).

CMake can simplify this task substantially. For now, refer to README.CMake.txt for further information.

**Todo** This section needs a major rework. Add a chapter "Building FLTK with CMake".

1.3.4.1 Compiling Programs with Standard Compilers

Under UNIX (and under Microsoft Windows when using the GNU development tools) you will probably need to tell the compiler where to find the header files. This is usually done using the `-I` option:

```c
cc++ -I/usr/local/include ...
```

**Note**

You need a C++ compiler to build FLTK. The commands given in this chapter are examples using `c++`. Please replace this command with the C++ compiler suitable for your system or use the `fltk-config` script as described below (this is recommended).
The `fltk-config` script included with FLTK can be used on systems with a Posix compliant shell, for instance Unix/Linux, macOS, Windows with MinGW, MSYS2, or Cygwin.

**Note**

`fltk-config` is not designed to work on Windows with Visual Studio compilers. If it works, then only by accident and this is undefined behavior.

`fltk-config --help` displays all available options.

`fltk-config` can be used to get the compiler and the options that are required by your compiler to build a program using the FLTK library:

```bash
fltk-config --cc
fltk-config --cxx
```

return the C and C++ compiler commands used to build FLTK.

```bash
c++ `fltk-config --cxxflags` ...
```

can be used to include the required compiler flags in the command line.

Similarly, when linking your application you will need to tell the compiler to use the FLTK library:

```bash
c++ ... -L/usr/local/lib -lfltk -lXext -lX11 ... -lm -ldl
```

Aside from the "fltk" library, there are also the following libraries:

- "fltk_forms" for the XForms compatibility classes (deprecated)
- "fltk_gl" for the OpenGL and GLUT classes
- "fltk_images" for the image file classes, Fl_Help_Dialog widget, and system icon support.

The libraries are named `fltk.lib`, `fltk_forms.lib`, `fltk_gl.lib`, and `fltk_images.lib` under Windows.

**Note**

The separate `fltk_cairo` library is no longer necessary since FLTK 1.4.0. However, this release of FLTK builds a dummy `fltk_cairo` library for backwards compatibility. You are advised to remove the usage of the `fltk_cairo` library from your build systems and tools. The `fltk_cairo` library will be removed in a future release.

As before, the `fltk-config` script can be used to get the options that are required by your linker:

```bash
c++ ... `fltk-config --ldflags`
```

The forms, GL, and images libraries are included with the "--use-foo" options, as follows:

```bash
c++ ... `fltk-config --use-forms --ldflags`
c++ ... `fltk-config --use-gl --ldflags`
c++ ... `fltk-config --use-cairo --ldflags`
c++ ... `fltk-config --use-forms --use-gl --use- images --ldflags`
```

The option "--use-cairo" may be used to build your program with Cairo libs if you use Cairo in your code. It does no longer include the `fltk_cairo` lib but all necessary Cairo compiler flags and Cairo libs, if and only if FLTK has been built with the optional Cairo support by configure or CMake.

Finally, you can use the `fltk-config` script to compile one or more source files as a FLTK program.

The following examples will create an executable named `filename` (or `filename.exe` under Windows) from a single source file:

```bash
fltk-config --compile filename.cxx
fltk-config --use-forms --compile filename.cpp
fltk-config --use-gl --compile filename.C
fltk-config --use-images --compile filename.cc
fltk-config --use-cairo --compile filename.cpp
fltk-config --use-forms --use-gl --use-images --compile filename.cpp
```

**Note**

`'fltk-config --compile'` accepts only a limited set of file extensions for C++ source files: `.cpp`, `.cxx`, `.cc`, and `.C` (capital 'C').
1.3.4.3 Compiling Multiple Source Files with 'fltk-config'

Before version 1.4.0 fltk-config accepted only a single source file and no additional compiler options or libraries. As of FLTK 1.4.0 it is possible to use additional compiler flags, more than one source file, and additional link libraries.

This is intended to be used for quick prototyping and not for production code development. It can be used to test compiler command options (like --Wall or --Wextra) or additional link libraries if these are required.

Building from more than one source file with flags and libraries can be achieved as follows:

```
fltk-config [USE-FLAGS] --compile MAIN [FLAGS] [SOURCES] [--link LFLAGS LIBS]
```

where

- arguments in [...] are optional
- USE-FLAGS are as described above, e.g. --use-images
- MAIN is the main C++ source file as documented above
- FLAGS are additional compiler flags
- SOURCES are additional source files or libraries
- --link is used to separate source files and flags from linker flags and libs
- LFLAGS are optional linker flags
- LIBS are additional libraries to link against

The final commandline is composed like this example:
```
$ fltk-config --compile main.cxx button.o --link -L/usr/include/cairo/ -lcairo
g++ {fltk-flags} -o main -Wextra main.cxx button.o x1.a {fltk-libs} -L/usr/include/cairo/ -lcairo
```

where {fltk-flags} are the compiler flags generated by fltk-config as before and {fltk-libs} are the usual linker flags and libraries. All optional parameters are used as-is, i.e. there is no syntax checking or special parsing except: the order of flags and source files is preserved (from the commandline) but all flags (--something) are positioned before all sources, i.e. arguments w/o leading dash ('-'). All compiler flags and libraries generated from the library build follow all options and source files given on the commandline, and finally everything after --link is appended.

1.3.4.4 Compiling Programs with Makefiles

The previous sections described how to use fltk-config to build a program from the command line, and this is very convenient for small test programs. But fltk-config can also be used to set the compiler and linker options as variables within a Makefile that can be used to build larger programs.

```
CXX = $(shell fltk-config --cxx)
DEBUG = -g
CXXFLAGS = $(shell fltk-config --use-gl --use-images --cxxflags ) -I.
LDFLAGS = $(shell fltk-config --use-gl --use-images --ldflags )
LDSTATIC = $(shell fltk-config --use-gl --use-images --ldstaticflags )
LINK = $(CXX)
TARGET = cube
OBJS = CubeMain.o CubeView.o CubeViewUI.o
SRCS = CubeMain.cxx CubeView.cxx CubeViewUI.cxx

.SUFFIXES: .o .cxx
%.o: %.cxx
  $(CXX) $(CXXFLAGS) $(DEBUG) -c $<
all: $(TARGET)
  $(LINK) -o $(TARGET) $(OBJS) $(LDSTATIC)
$(TARGET): $(OBJS)
  CubeMain.o: CubeMain.cxx CubeViewUI.h
  CubeView.o: CubeView.cxx CubeViewUI.h
  CubeViewUI.o: CubeViewUI.cxx CubeViewUI.h
clean: $(TARGET) $(OBJS)
  rm -f *.o 2>&1 | /dev/null
  rm -f $(TARGET) 2>&1 | /dev/null
```

Generated by Doxygen
1.3.4.5 Compiling Programs with Microsoft Visual C++

In Visual C++ you will need to tell the compiler where to find the FLTK header files. This can be done by selecting "Settings" from the "Project" menu and then changing the "Preprocessor" settings under the "C/C++" tab.

You will also need to add the following libraries to the Linker settings:

- `fltk.lib` or `fltkd.lib`, the main FLTK library (postfix 'd' = Debug)
- all FLTK libraries your program requires (`fltk_gl`, `fltk_images`, ...)
- additional libraries like `libpng.lib`, `libjpeg.lib`, etc.
- the Windows Common Controls (`comctl32.lib`) and
- the GDIplus library if used to build FLTK (`gdiplus.lib`) and
- the Windows Socket (`ws2_32.lib`) libraries.

**Note**

There's a Linker setting "Additional Library Directories" or similar; the exact name depends on the Visual Studio version you're using. You can and **should** use this to simplify adding the libraries above. If you set this to the FLTK library path you can just use the library **names** and don't need to use the full paths to all libraries.

You must also define `_WIN32` if the compiler doesn't do this. Currently all known Windows compilers define `_WIN32` - unless you use Cygwin (that's correct, you must not define `_WIN32` if you use Cygwin).

More information can be found in README.Windows.txt.

You can build your Microsoft Windows applications as Console or Desktop applications. If you want to use the standard `C main()` function as the entry point, FLTK includes a `WinMain()` function that will call your `main()` function for you.

## 1.4 Common Widgets and Attributes

This chapter describes many of the widgets that are provided with FLTK and covers how to query and set the standard attributes.

### 1.4.1 Buttons

FLTK provides many types of buttons:

- **Fl_Button** - A standard push button.

- **Fl_Check_Button** - A button with a check box.

- **Fl_Light_Button** - A push button with a light.

- **Fl_Repeat_Button** - A push button that repeats when held.
• **FL_Return_Button** - A push button that is activated by the *Enter* key.

• **FL_Round_Button** - A button with a radio circle.

![Figure 1.2 FLTK Button Widgets](image)

All of these buttons just need the corresponding `<FL/Fl_xyz_Button.H>` header file. The constructor takes the bounding box of the button and optionally a label string:

```cpp
Fl_Button *button = new Fl_Button(x, y, width, height, "label");
Fl_Light_Button *lbutton = new Fl_Light_Button(x, y, width, height);
Fl_Round_Button *rbutton = new Fl_Round_Button(x, y, width, height, "label");
```

Each button has an associated `type()` which allows it to behave as a push button, toggle button, or radio button:

```cpp
button->type(FL_NORMAL_BUTTON);
lbutton->type(FL_TOGGLE_BUTTON);
rbutton->type(FL_RADIO_BUTTON);
```

For toggle and radio buttons, the `value()` method returns the current button state (0 = off, 1 = on). The `set()` and `clear()` methods can be used on toggle buttons to turn a toggle button on or off, respectively. Radio buttons can be turned on with the `setonly()` method; this will also turn off other radio buttons in the same group.

### 1.4.2 Text

FLTK provides several text widgets for displaying and receiving text:

• **Fl_Input** - A one-line text input field.

• **Fl_Output** - A one-line text output field.

• **Fl_Multiline_Input** - A multi-line text input field.

• **Fl_Multiline_Output** - A multi-line text output field.

• **Fl_Text_Display** - A multi-line text display widget.

• **Fl_Text_Editor** - A multi-line text editing widget.

• **Fl_Help_View** - A HTML text display widget.

The `Fl_Output` and `Fl_Multiline_Output` widgets allow the user to copy text from the output field but not change it.

The `value()` method is used to get or set the string that is displayed:

```cpp
Fl_Input *input = new Fl_Input(x, y, width, height, "label");
input->value("Now is the time for all good men...");
```

The string is copied to the widget's own storage when you set the `value()` of the widget.

The `Fl_Text_Display` and `Fl_Text_Editor` widgets use an associated `Fl_Text_Buffer` class for the value, instead of a simple string.
1.4.3 Valuators

Unlike text widgets, valuators keep track of numbers instead of strings. FLTK provides the following valuators:

- **FL.Counter** - A widget with arrow buttons that shows the current value.
- **FL.Dial** - A round knob.
- **FL.Roller** - An SGI-like dolly widget.
- **FL.Scrollbar** - A standard scrollbar widget.
- **FL.Slider** - A scrollbar with a knob.
- **FL.Value_Slider** - A slider that shows the current value.

The `value()` method gets and sets the current value of the widget. The `minimum()` and `maximum()` methods set the range of values that are reported by the widget.
1.4.4 Groups

The Fl_Group widget class is used as a general purpose "container" widget. Besides grouping radio buttons, the groups are used to encapsulate windows, tabs, and scrolled windows. The following group classes are available with FLTK:

- Fl_Double_Window - A double-buffered window on the screen.
- Fl_Gl_Window - An OpenGL window on the screen.
- Fl_Group - The base container class; can be used to group any widgets together.
- Fl_Pack - A collection of widgets that are packed into the group area.
- Fl_Scroll - A scrolled window area.
- Fl_Tabs - Displays child widgets as tabs.
- Fl_Tile - A tiled window area.
- Fl_Window - A window on the screen.
- Fl_Wizard - Displays one group of widgets at a time.

1.4.5 Setting the Size and Position of Widgets

The size and position of widgets is usually set when you create them. You can access them with the x(), y(), w(), and h() methods.

You can change the size and position by using the position(), resize(), and size() methods:

```
button->position(x, y);
group->resize(x, y, width, height);
window->size(width, height);
```

If you change a widget's size or position after it is displayed you will have to call redraw() on the widget's parent.
1.4 Common Widgets and Attributes

1.4.6 Colors

FLTK stores the colors of widgets as an 32-bit unsigned number that is either an index into a color palette of 256 colors or a 24-bit RGB color. The color palette is not the X or MS Windows colormap, but instead is an internal table with fixed contents.

See the Colors section of Drawing Things in FLTK for implementation details.

There are symbols for naming some of the more common colors:

- FL_BLACK
- FL_RED
- FL_GREEN
- FL_YELLOW
- FL_BLUE
- FL_MAGENTA
- FL_CYAN
- FL_WHITE

Other symbols are used as the default colors for all FLTK widgets.

- FL_FOREGROUND_COLOR
- FL_BACKGROUND_COLOR
- FL_INACTIVE_COLOR
- FL_SELECTION_COLOR

The full list of named color values can be found in FLTK Enumerations.

A color value can be created from its RGB components by using the `fl_rgb_color()` function, and decomposed again with `Fl::get_color()`:

```cpp
Fl_Color c = fl_rgb_color(85, 170, 255); // RGB to Fl_Color
Fl::get_color(c, r, g, b); // Fl_Color to RGB
```

The widget color is set using the `color()` method:

```cpp
button->color(FL_RED); // set color using named value
```

Similarly, the label color is set using the `labelcolor()` method:

```cpp
button->labelcolor(FL_WHITE);
```

The Fl_Color encoding maps to a 32-bit unsigned integer representing RGBI, so it is also possible to specify a color using a hex constant as a color map index:

```cpp
button->color(0x000000ff); // colormap index #255 (FL_WHITE)
```

or specify a color using a hex constant for the RGB components:

```cpp
button->color(0xff000000); // RGB: red
button->color(0x00ff0000); // RGB: green
button->color(0x0000ff00); // RGB: blue
button->color(0xffffff00); // RGB: white
```

Note

If TrueColor is not available, any RGB colors will be set to the nearest entry in the colormap.
1.4.7 Box Types

The type Fl_Boxtype stored and returned in Fl_Widget::box() is an enumeration defined in Enumerations.H.

These are the standard box types included with FLTK:

![Figure 1.4 FLTK Standard Box Types](image)

**FL_NO_BOX** means nothing is drawn at all, so whatever is already on the screen remains. The **FL_..._FRAME** types only draw their edges, leaving the interior unchanged. The blue color in the image above is the area that is not drawn by the frame types.

### 1.4.7.1 Making Your Own Boxtypes

You can define your own boxtypes by making a small function that draws the box and adding it to the table of boxtypes.

#### The Drawing Function

The drawing function is passed the bounding box and background color for the widget:

```c
void xyz_draw(int x, int y, int w, int h, Fl_Color c) {
    ...
}
```

A simple drawing function might fill a rectangle with the given color and then draw a black outline:

```c
void xyz_draw(int x, int y, int w, int h, Fl_Color c) {
    fl_color(c);
    fl_rectf(x, y, w, h);
    fl_color(FL_BLACK);
    fl_rect(x, y, w, h);
}
```

**Fl_Boxtype fl_down(Fl_Boxtype b)**
fl_down() returns the “pressed” or “down” version of a box. If no “down” version of a given box exists, the behavior of this function is undefined and some random box or frame is returned. See Drawing Functions for more details.

Fl_Boxtype fl_frame(Fl_Boxtype b)

fl_frame() returns the unfilled, frame-only version of a box. If no frame version of a given box exists, the behavior of this function is undefined and some random box or frame is returned. See Drawing Functions for more details.

Fl_Boxtype fl_box(Fl_Boxtype b)

fl_box() returns the filled version of a frame. If no filled version of a given frame exists, the behavior of this function is undefined and some random box or frame is returned. See Drawing Functions for more details.

Adding Your Box Type

The Fl::set_boxtype() method adds or replaces the specified box type:

```c
#define XYZ_BOX FL_FREE_BOXTYPE
Fl::set_boxtype(XYZ_BOX, xyz_draw, 1, 1, 2, 2);
```

The last 4 arguments to Fl::set_boxtype() are the offsets for the x, y, width, and height values that should be subtracted when drawing the label inside the box.

A complete box design contains four box types in this order: a filled, neutral box (UP_BOX), a filled, depressed box (DOWN_BOX), and the same as outlines only (UP_FRAME and DOWN_FRAME). The function fl_down(Fl_Boxtype) expects the neutral design on a boxtype with a numerical value evenly dividable by two. fl_frame(Fl_Boxtype) expects the UP_BOX design at a value dividable by four.

1.4.8 Labels and Label Types

The label(), align(), labelfont(), labelsize(), labeltype(), image(), and deimage() methods control the labeling of widgets.
The label() method sets the string that is displayed for the label. Symbols can be included with the label string by escaping them using the "@" symbol - "@@" displays a single at sign. These are the available symbols:

![FLTK label symbols](image)

The @ sign may also be followed by the following optional "formatting" characters, in this order:

- '#' forces square scaling, rather than distortion to the widget's shape.
- +[1-9] or -[1-9] tweaks the scaling a little bigger or smaller.
- '$' flips the symbol horizontally, '%' flips it vertically.
- [0-9] - rotates by a multiple of 45 degrees. '5' and '6' do no rotation while the others point in the direction of that key on a numeric keypad. '0', followed by four more digits rotates the symbol by that amount in degrees.

Thus, to show a very large arrow pointing downward you would use the label string "@+92->".

Symbols and text can be combined in a label, however the symbol must be at the beginning and/or at the end of the text. If the text spans multiple lines, the symbol or symbols will scale up to match the height of all the lines.
align()

The `align()` method positions the label. The following constants are defined and may be OR'd together as needed:

- `FL_ALIGN_CENTER` - center the label in the widget.
- `FL_ALIGN_TOP` - align the label at the top of the widget.
- `FL_ALIGN_BOTTOM` - align the label at the bottom of the widget.
- `FL_ALIGN_LEFT` - align the label to the left of the widget.
- `FL_ALIGN_RIGHT` - align the label to the right of the widget.
- `FL_ALIGN_LEFT_TOP` - The label appears to the left of the widget, aligned at the top. Outside labels only.
- `FL_ALIGN_RIGHT_TOP` - The label appears to the right of the widget, aligned at the top. Outside labels only.
- `FL_ALIGN_LEFT_BOTTOM` - The label appears to the left of the widget, aligned at the bottom. Outside labels only.
- `FL_ALIGN_RIGHT_BOTTOM` - The label appears to the right of the widget, aligned at the bottom. Outside labels only.
- `FL_ALIGN_INSIDE` - align the label inside the widget.
- `FL_ALIGN_CLIP` - clip the label to the widget's bounding box.
- `FL_ALIGN_WRAP` - wrap the label text as needed.
- `FL_ALIGN_TEXT_OVER_IMAGE` - show the label text over the image.
- `FL_ALIGN_IMAGE_OVER_TEXT` - show the label image over the text (default).
- `FL_ALIGN_IMAGE_NEXT_TO_TEXT` - The image will appear to the left of the text.
- `FL_ALIGN_TEXT_NEXT_TO_IMAGE` - The image will appear to the right of the text.
- `FL_ALIGN_IMAGE_BACKDROP` - The image will be used as a background for the widget.
labeltype()

The `labeltype()` method sets the type of the label. The following standard label types are included:

- `FL_NORMAL_LABEL` - draws the text.
- `FL_NO_LABEL` - does nothing.
- `FL_SHADOW_LABEL` - draws a drop shadow under the text.
- `FL_ENGRAVED_LABEL` - draws edges as though the text is engraved.
- `FL_EMBOSSED_LABEL` - draws edges as though the text is raised.
- `FL_ICON_LABEL` - draws the icon (`Fl_Image`) associated with the text.
- `FL_IMAGE_LABEL` - draws the image (`Fl_Image`) associated with the text.
- `FL_MULTI_LABEL` - draws multiple parts side by side, see `Fl_Multi_Label`.

Note

Some of these labeltypes are no longer necessary for normal widgets. Widgets allow for an image and a text side by side, depending on the widget's `align()` flag. `FL_MULTI_LABEL` was designed to be used with `Fl_MENU_ITEM`'s to support icons or small images, typically left of the menu text.

As of this writing (FLTK 1.4.0, Sep 2017) `Fl_MENU_ITEMS` support only one label part (text or image), but using `Fl_Multi_Label` as the label can extend this to more than one part.

See also

- class `Fl_Multi_Label`, `Fl_Widget::align()`

image() and deimage()

The `image()` and `deimage()` methods set an image that will be displayed with the widget. The `deimage()` method sets the image that is shown when the widget is inactive, while the `image()` method sets the image that is shown when the widget is active.

To make an image you use a subclass of `Fl_Image`.

Making Your Own Label Types

Label types are actually indexes into a table of functions that draw them. The primary purpose of this is to use this to draw the labels in ways inaccessible through the `fl_font()` mechanism (e.g. `FL_ENGRAVED_LABEL`) or with program-generated letters or symbology.
Label Type Functions

To setup your own label type you will need to write two functions: one to draw and one to measure the label. The draw function is called with a pointer to a Fl_Label structure containing the label information, the bounding box for the label, and the label alignment:

```c
void xyz_draw(const Fl_Label *label, int x, int y, int w, int h, Fl_Align align) {
    ...
}
```

The label should be drawn inside this bounding box, even if FL_ALIGN_INSIDE is not enabled. The function is not called if the label value is NULL.

The measure function is called with a pointer to a Fl_Label structure and references to the width and height:

```c
void xyz_measure(const Fl_Label *label, int &w, int &h) {
    ...
}
```

The function should measure the size of the label and set \( w \) and \( h \) to the size it will occupy.

Adding Your Label Type

The Fl::set_labeltype() method creates a label type using your draw and measure functions:

```c
#define XYZ_LABEL FL_FREE_LABELTYPE
Fl::set_labeltype(XYZ_LABEL, xyz_draw, xyz_measure);
```

The label type number \( n \) can be any integer value starting at the constant FL_FREE_LABELTYPE. Once you have added the label type you can use the labeltype() method to select your label type.

The Fl::set_labeltype() method can also be used to overload an existing label type such as FL_NORMAL_LABEL.

Making your own symbols

It is also possible to define your own drawings and add them to the symbol list, so they can be rendered as part of any label.

To create a new symbol, you implement a drawing function `void drawit(Fl_Color c)` which typically uses the functions described in Drawing Complex Shapes to generate a vector shape inside a two-by-two units sized box around the origin. This function is then linked into the symbols table using fl_add_symbol():

```c
int fl_add_symbol(const char *name, void (*drawit)(Fl_Color), int scalable)
```

`name` is the name of the symbol without the "@"; `scalable` must be set to 1 if the symbol is generated using scalable vector drawing functions.

```c
int fl_draw_symbol(const char *name, int x, int y, int w, int h, Fl_Color col)
```

This function draws a named symbol fitting the given rectangle.
1.4.9 Callbacks

Callbacks are functions that are called when the value of a widget changes. A callback function is sent a `Fl_Widget` pointer of the widget that changed and a pointer to data that you provide:

```c
void xyz_callback(Fl_Widget *w, void *data) {
...
}
```

The `callback()` method sets the callback function for a widget. You can optionally pass a pointer to some data needed for the callback:

```c
int xyz_data;
button->callback(xyz_callback, &xyz_data);
```

Note

You cannot delete a widget inside a callback, as the widget may still be accessed by FLTK after your callback is completed. Instead, use the `Fl::delete_widget()` method to mark your widget for deletion when it is safe to do so.

Many programmers new to FLTK or C++ try to use a non-static class method instead of a static class method or function for their callback. Since callbacks are done outside a C++ class, the `this` pointer is not initialized for class methods.

To work around this problem, define a static method in your class that accepts a pointer to the class, and then have the static method call the class method(s) as needed. The data pointer you provide to the `callback()` method of the widget can be a pointer to the instance of your class.

```c
class Foo {
  void my_callback(Fl_Widget *w);
  static void my_static_callback(Fl_Widget *w, void *f) {
    ((Foo *)f)->my_callback(w);
  }
  ...
  ...
  w->callback(my_static_callback, (void *)this);
}
```

In an effort to make callbacks easier, more flexible, and type safe, FLTK provides three groups of macros that generate the code needed to call class methods directly with up to five custom parameters.

- `FL_FUNCTION_CALLBACK_#(WIDGET, FUNCTION, ...)` creates code for callbacks to functions and static class methods with up to five arguments. The `#` must be replaced by the number of callback arguments.
- `FL_METHOD_CALLBACK_#(WIDGET, CLASS, SELF, METH, ...)` creates code for callbacks to arbitrary public class methods
- `FL_INLINE_CALLBACK_#(WIDGET, ..., FUNCTION_BODY)` creates code for callback functions that are very close to (almost in the same line) the widget creation code, similar to lambda function in C++11. The last argument of this macro is the callback code.

The syntax is a bit unconventional, but the resulting code is flexible and needs no additional maintenance. It is also C++98 compatible. For example:

```c
#include <FL/fl_callback_macros.H>

... Fl_String *str = new Fl_String("FLTK");
Fl_Button *btn = new Fl_Button(10, 10, 100, 100);
FL_METHOD_CALLBACK_2(btn, Fl_String, str, insert, int, 2, const char *, "...");
...
Fl_Button *inline_cb_btn_2 = new Fl_Button(390, 60, 180, 25, "2 args");
FL_INLINE_CALLBACK_2( inline_cb_btn_2, const char *, text, "FLTK", int, number, 2,
  { fl_message("We received the message %s with %d!", text, number); })
...
```

See also

- `Fl_Widget::callback(Fl_Callback*, void*)`
- `FL_FUNCTION_CALLBACK_3`
- `FL_METHOD_CALLBACK_1`
- `FL_INLINE_CALLBACK_2`
1.4.10 When and Reason

Normally callbacks are performed only when the value of the widget changes. You can change this using the `Fl_Widget::when()` method:

```cpp
button->when(FL_WHEN_NEVER);
button->when(FL_WHEN_CHANGED);
button->when(FL_WHEN_RELEASE);
button->when(FL_WHEN_RELEASE_ALWAYS);
button->when(FL_WHEN_ENTER_KEY);
button->when(FL_WHEN_ENTER_KEY_ALWAYS);
button->when(FL_WHEN_CHANGED | FL_WHEN_NOT_CHANGED);
```

Within the callback, you can query why the callback was called using `Fl::callback_reason()`. For example, setting

```cpp
myInput->when(FL_WHEN_RELEASE | FL_WHEN_CHANGED)
```

for a text input field may return `FL_REASON_LOST_FOCUS` or `FL_REASON_CHANGED` as a callback reason.

1.4.11 Shortcuts

Shortcuts are key sequences that activate widgets such as buttons or menu items. The `shortcut()` method sets the shortcut for a widget:

```cpp
button->shortcut(FL_Enter);
button->shortcut(FL_SHIFT + 'b');
button->shortcut(FL_CTRL + 'b');
button->shortcut(FL_ALT + 'b');
button->shortcut(FL_CTRL + FL_ALT + 'b');
button->shortcut(0); // no shortcut
```

The shortcut value is the key event value - the ASCII value or one of the special keys described in `Fl::event_key()` Values combined with any modifiers like `Shift`, `Alt`, and `Control`.

1.5 Coordinates and Layout Widgets

This chapter describes the coordinate systems that apply when positioning widgets manually, and some of the basics of FLTK layout widgets that are used to position widgets automatically.

1.5.1 The Widget Coordinate System

All widgets have constructors with `x` and `y` parameters to let the programmer specify the desired initial position of the top left corner during explicit manual layout within `Fl_Window` and `Fl_Group` container widgets.

This position is always relative to the enclosing `Fl_Window`, which is usually, but not always, the top-level application window, or a free-floating pop-up dialog window. In some cases it could also be a subwindow embedded in a higher-level window, as shown in the figure below.

![Figure 1.7 FLTK coordinate system](image)

`Fl_Window TL(0, 0, 250, 100)`
`x, y relative to main window`

`Fl_Group BL(100, 250, 100)`
`x, y relative to main window`

`Fl_Box tl(25, 50, 200, 40)`
`x, y relative to TL window`

`Fl_Box bl(25, 150, 200, 40)`
`x, y relative to main window`

`Fl_Window TR(250, 0, 250, 100)`
`x, y relative to main window`

`Fl_Window BR(250, 100, 250, 100)`
`x, y relative to main window`

`Fl_Box tr(25, 50, 200, 40)`
`x, y relative to TR window`

`Fl_Box br(25, 150, 200, 40)`
`x, y relative to BR window`

Figure 1.7 FLTK coordinate system

Generated by Doxygen
The positions of the TL and BR sub-windows and the TR and BL groups are all relative to the top-left corner of the main window. The positions of the boxes inside the TR and BL groups are also relative to the main window, but the boxes inside the TL and BR sub-windows are positioned relative to the enclosing sub-window.

In other words, the widget hierarchy and positions can be summarized as:

```
Fl_Window main window
Fl_Window TL subwindow # x, y relative to main window
   Fl_Box tl box # x, y relative to TL subwindow
Fl_Window BR subwindow # x, y relative to main window
   Fl_Box br box # x, y relative to BR subwindow
Fl_Group TR group # x, y relative to main window
   Fl_Box tr box # x, y relative to main window
Fl_Group BL group # x, y relative to main window
   Fl_Box bl box # x, y relative to main window
```

1.5.2 Layout and Container Widgets

There are four main groups of widgets derived from Fl_Group for a range of different purposes.

The first group are composite widgets that each contain a fixed set of components that work together for a specific purpose, rather than layout widgets as such, and are not discussed here.

The second group are basically containers offering the same manual layout features as Fl_Group, as described above, but which add one new capability. These widgets are Fl_Scroll, Fl_Tabs and Fl_Wizard.

The third group are layout managers that relocate and resize the child widgets added to them in order to satisfy a particular layout algorithm. These widgets are Fl_Flex, Fl_Grid, Fl_Pack, and Fl_Tile.

The final group consists of Fl_Window and its derivatives. Their special capability is that they can be top-level application windows and dialogs that interface with the operating system window manager, but can also be embedded within other windows and groups as shown in the example above. Note that the window manager may impose its own constraints on the position of top-level windows, and the x and y position parameters may be treated as hints, or even ignored. The Fl_Window class has an extra constructor that omits them.

Descriptions of layout and container widgets follow in alphabetical order.

1.5.2.1 The Fl_Flex Layout Widget

The Fl_Flex widget allows the layout of its direct children as a single row or column. If its type() is set to give the row or horizontal layout, the children are all resized to have the same height as the Fl_Flex and are moved next to each other. If set to give the column or vertical layout, the children are all resized to have the same width as the Fl_Flex and are then stacked below each other.

Widget positions (x, y) need not be given by the user because widgets are positioned inside the Fl_Flex container in the order of its children. Widget sizes can be set to (0, 0) as in Fl_Pack since they are calculated by Fl_Flex.

This is similar to Fl_Pack described below and Fl_Flex is designed to act as a drop-in replacement of Fl_Pack with some minor differences.

Other than Fl_Pack the Fl_Flex widget does not resize itself but resizes its children to fill the entire space of the Fl_Flex container. Single children of Fl_Flex can be set to fixed sizes to inhibit this resizing behavior. In this case the remaining space is distributed to all non-fixed widgets.

Fl_Flex widgets can be nested inside each other and with Fl_Grid in any combination.

The name Fl_Flex was inspired by the CSS ‘flex’ container.

Figure 1.8 Simple Fl_Flex Layout

Fl_Flex was added in FLTK 1.4.0.
1.5.2.2 The Fl_Grid Layout Widget

Fl_Grid is the most flexible layout container in FLTK 1.4. It is based on a flexible grid of cells that can be assigned one widget per cell which is the anchor of the widget. Widgets can span multiple rows and columns and the cells can constitute a sparse matrix. Widgets can be aligned inside their cells in several ways (left, right, top, bottom) and can stretch horizontally, vertically, or both, i.e. fill the entire cell.

Widget positions (x, y) need not be given by the user because widgets are assigned to a particular grid cell by row and column number. Widget sizes can be given as their minimal sizes and will be resized appropriately depending on the free space.

Optional margins around all cells inside the widget border and gaps between rows and cells make the layout even more flexible.

The Fl_Grid widget should be designed with a grid (matrix) and its minimal size in mind. It is designed to enlarge cells and widgets in a flexible way when the Fl_Grid widget itself is created or resized.

Additional free space inside the Fl_Grid container is distributed to widgets by considering minimal row heights, column widths, sizes of widgets, and row and column weights. These weights are used to distribute the free space proportionally according to the row and column weights.

Fl_Grid widgets can be nested inside each other and with Fl_Flex and other subclasses of Fl_Group in any combination.

Note

We don't recommend to use Fl_Pack as child widgets although this may work as well.

The name Fl_Grid was inspired by the CSS ‘grid’ container but it has some properties in common with HTML <table> containers as well, for instance row and column spanning.

Figure 1.9 Simple Fl_Grid Layout

Fl_Grid was added in FLTK 1.4.0.
1.5.2.3 The Fl_Pack Layout Widget

The Fl_Pack widget allows the layout of its direct children as a single row, or column. If its type() is set to give the row or horizontal layout, the children are all resized to have the same height as the Fl_Pack and are moved next to each other. If set to give the column or vertical layout, the children are all resized to have the same width as the Fl_Pack and are then stacked below each other. The Fl_Pack then resizes itself to shrink-wrap itself around all of the children.

Fl_Pack widgets are often used inside an Fl_Scroll, as shown in the diagram below, to avoid having to deal with tricky resize behavior when used with nested widgets.

Since FLTK 1.4.0 Fl_Flex (described above) can in many cases be used as a drop-in replacement for Fl_Pack if this "shrink-wrap" behavior is not required. Note that the Fl_Pack layout algorithm can cause some issues because its widget size can change depending on its children and particularly because this is done late, i.e. during draw() and not as usual during resize of the window.

Note

We recommend that developers evaluate whether using Fl_Flex or Fl_Grid instead of Fl_Pack can be a better solution with more predictable and reliable resizing behavior of the overall program layout.

1.5.2.4 The Fl_Scroll Container Widget

The Fl_Scroll container widget can hold an assortment of widgets that may extend beyond its own width and height, in which case horizontal and/or vertical scrollbars may appear automatically so that you can scroll and view the entire contents.
1.5.2.5 The Fl_Tabs Container Widget

The Fl_Tabs widget provides a front-to-back stack of individual panels which usually contain Fl_Group widgets and their children. The user can switch between panels by clicking on the small tabs that protrude from the panels. The appearance of each tab is determined by the child widget’s label and related attributes.

![Fl_Tabs container widget](image)

Figure 1.12 Fl_Tabs container widget

1.5.2.6 The Fl_Tile Layout Widget

The Fl_Tile widget allows the user to resize one or more of its children by dragging on the border between adjacent child widgets. However, the programmer must first explicitly layout the child widgets so that their borders exactly fill the width and height of the Fl_Tile without having any gaps between them, or at the edges. Some care is needed when initially positioning the children and setting the resizable() widget within the Fl_Tile to prevent squeezing a child to have a zero width or height. For more information see the Fl_Tile widget manual page, and How Does Resizing Work?.

![Fl_Tile layout widget](image)

Figure 1.13 The Fl_Tile layout widget

1.5.2.7 The Fl_Wizard Container Widget

The Fl_Wizard widget derives from the Fl_Tabs class, but instead of having tabs that the user can click to select the corresponding panel, the programmer uses the prev(), next() or value() methods to show the appropriate panel. For example, the user might be able to click on "Next" and "Prev" navigation buttons or keys, as shown below.

![Fl_Wizard container widget](image)

Figure 1.14 Fl_Wizard container widget
1.6 How Does Resizing Work?

This chapter describes the basic mechanism behind the creation of resizable user interface elements in FLTK.

FLTK uses a simple, but very versatile system to resize even the most complex dialogs and interfaces. The resizing is implemented within the `Fl_Group` widget, and the exact resizing behavior of that group is determined by its `resizable()` attribute.

1.6.1 Resizing can be disabled

Summary:
```
group = new Fl_Group(xg, yg, wg, hg, "No Resizing");
child1 = new Fl_Box(xb, yb, wb, hb, "B"); // or other widget type
...
group->resizable((Fl_Widget*)0); // no resizing
group->end()
```

The `resizable` may be set to the NULL pointer, which means that the group will not resize. Note that this is the default behavior for `Fl_Window` and `Fl_Pack` derived widgets, and therefore the programmer must explicitly set the window's `resizable` attribute if they want to allow the window to be resized.

1.6.2 Resizing can be simple

Summary:
```
group = new Fl_Group(xg, yg, wg, hg, "Simple Resizing");
child1 = new Fl_Box(xb, yb, wb, hb, "B"); // or other widget type
...
group->resizable(group); // simple proportional resizing
group->end()
```

The `resizable` may be set to the group itself, which means that all widgets within the group will resize as the group itself is resized. This is the default behavior for `Fl_Group` widgets, and is shown in the diagram below.

If the group is stretched horizontally, the widths of the widgets within the group are adjusted proportionally. The same is true for vertical resizing.

![Figure 1.15 Proportional resizing example](image)

Figure 1.15 Proportional resizing example
1.6 How Does Resizing Work?

1.6.3 Resizing can be complex

Summary:

```cpp
  group = new Fl_Group(xg, yg, wg, hg, "Complex Resizing");
  child1 = new Fl_Box(xb, yb, wb, hb, "B"); // or other widget type
  ...
  group->resizable(child1); // complex resizing
  group->end()
```

It is when the group's `resizable` attribute is set to one of the group's child widgets, that things become really interesting.

In the diagram below, imagine vertical lines extending from the left and right sides of the yellow widget marked "resizable", and horizontal lines extending from the top and bottom sides. Exactly which widgets resize, and by how much, is determined by which ones lie completely or partially within this cross.

The widgets marked B, C, J, K and M clearly lie completely or partially within the vertical part of the cross; the widgets marked E, F, G, H and N lie completely or partially within the horizontal part of the cross; and the widgets marked A, D, I and L do not overlap with the cross at all. The resizing behavior is as follows:

- the width and height of the `resizable` widget increase to match the change in the width and height of the group widget as it is stretched;

- the widths of those widgets that overlap with the vertical part of the cross increase proportionally as the width of the group widget increases, but their heights remain unchanged, i.e. the widgets marked B, C, J, K and M;

- the heights of those widgets that overlap with the horizontal part of the cross increase proportionally as the height of the group widget increases, but their widths remain unchanged, i.e. the widgets marked E, F, G, H and N;

- the widths and heights of the remaining widgets stay the same, i.e. the widgets marked A, D, I and L stay the same size.

![Figure 1.16 Complex resizing example](image-url)
1.6.4 Practical examples

Why is this so powerful, you may ask. Well, every widget group can have a completely independent resizing strategy. By replacing one or more of the group's "normal" child widgets with another group widget where all of the above rules can be applied again, it is possible to create a hierarchy of group widgets with very complex layouts and resizing behavior.

Consider a simple dialog box, consisting of an icon box and a message area on the top and a button at the bottom right: which widget should be the resizable one?

Setting the resizable to be the icon box won't give us what we want:

![Figure 1.17 Resizing dialog example (a)](image)

The message text area would be the logical choice so that the user can expand the dialog to see if there is more of an explanation below the short error message. This results in the behavior shown in the diagram below.

![Figure 1.18 Resizing dialog example (b)](image)

The result is close to what we want, but not quite: the text area will fully resize, the "!" icon box will resize vertically but not horizontally, which we can live with, but the "Darn!" button will - wait a minute - resize horizontally?

That's ugly. How do we stop that from happening? Simple: put it in its own group and set the resizable to an invisible box widget, as shown in the diagram below.
1.6 How Does Resizing Work?

Now the invisible box, shown as "R", takes all of the horizontal resizing and the "Darn!" box will stay as it is. Here's the skeleton code:

```c
dialog = new Fl_Window(300, 100);
icon = new Fl_Box(0, 0, 50, 50, "!");
text = new Fl_Box(50, 0, 250, 40, "Out of Memory Error");
btns = new Fl_Group(50, 50, 250, 50); // parent group
darn = new Fl_Button(200, 50, 100, 50, "Darn!");
R = new Fl_Box(50, 50, 150, 50); // "invisible" box
R->hide(); // make sure it's invisible
btns->resizable(R); // make *R* parent group resizable
dialog->resizable(text);
dialog->end();
```

Imagine instead that you have a group that has a button, an input field, another button and a second input field, all next to each other, and you want the input fields to resize equally, but not the buttons. How could you achieve this?

Setting either of the input fields to be the `resizable` leaves the other one fixed, as shown below:

![Figure 1.20 Resizing input fields example (b)](image1)

The answer is to leave the `resizable` of the group set to itself, and to create two equal size subgroups, each of which will resize equally. Add a button and input field to each subgroup, and set each subgroup's `resizable` to the input field, as shown below. Tada!
In FLTK it is possible to solve almost any layout and resizing problem by introducing an invisible box into a group, or an extra group into the widget hierarchy. It might take some thought to achieve exactly what you want and sometimes it is necessary to introduce parallel hierarchies in order to get widgets in different groups to resize together.

Imagine you have a group containing three widgets in a row, and you want the widget in the middle to stay the same size when the group is stretched and the ones on either side and the padding between them to resize symmetrically. As described earlier, the default resizing behavior for a group results in proportional resizing of the child widgets (and also of the margins and padding between them) as shown below, which is clearly not what you want.

![Figure 1.22 Resizing a row of widgets (a)](image)

Simply adding a group around A and B and setting its `resizable` to A, as in the previous btn-input example, will mean that B stays the same size, but the other widgets won't resize symmetrically, so what else is needed? It isn't immediately obvious how to solve this problem, even for experienced FLTK users. This is possibly because users are generally advised to design widgets so that they don't overlap.

Albrecht Schlosser proposed an innovative technique that involves an invisible box that deliberately overlaps others to achieve the desired behavior. For the current example, this means inserting two new groups into the existing group and adding a hidden resizable widget.

The first group, shown in red below, extends from the left edge of the parent group to the middle of the gap between boxes B and C on the right. This first group contains boxes A and B, where A is the first group's `resizable` attribute.

The second group, shown in blue, extends from the right edge of the first group to the right edge of the parent group. This second group contains box C, where C is the second group's `resizable`.

The extra box widget is added to the parent group and is set as the group's `resizable`. The three `resizable` widgets are shown in yellow.

The clever bit is that this extra box widget is not horizontally aligned with any of the existing groups and widgets in the usual way, but instead overlaps the right and left parts of the two new groups by the same small amount, which means that its midpoint is aligned with the edge between the groups.

Note that, for clarity, the height of the original group has been increased to allow space for the additional annotation and to highlight the extra resizable box in the extra space at the bottom of the group. This is fine for the horizontal-only resizing shown here, but means that widgets A, B and C will never change height because the extra resizable box does not overlap them vertically. Only the padding below them will be resized.

![Figure 1.23 Resizing a row of widgets (b)](image)
In a real application, you probably want to allow widgets A, B and C to resize vertically while the height of any padding or widgets above or below remains fixed, so the extra resizable box has to lie within the height of widgets A, B and C. Obviously after calling `hide()` on the box it is no longer visible, and may therefore be the same height as the other widgets, or a fraction of the height, as shown below.

![Figure 1.24 Resizing a row of widgets (c)](image)

To summarize the key points of the new technique:

- The new resizable widget must overlap the widgets on each side by exactly the same amount.
- The width of the new resizable widget is not fixed, but should probably be a relatively small value to avoid potential problems.
- The total width of the two new groups must equal the width of the existing group and there can be no offsets or gaps between them because margins and gaps will affect the resizing behavior.
- The same principles apply to vertical resizing.

### 1.7 Designing a Simple Text Editor

This chapter guides you through the design of a simple FLTK-based text editor.

The complete source code for our text editor can be found in the `test/editor.cxx` file.

The tutorial comprises multiple chapters, and you can activate the relevant code by adjusting the `TUTORIAL_\_\_CHAPTER` macro at the top of the source file to match the chapter number.

Each chapter builds on the previous one. The documentation, as well as the source code, can be read sequentially, maintaining a consistent program structure while introducing additional features step by step.

**Note**

The tutorial uses several global variables for brevity. Additionally, the order of code blocks is rather uncommon but helps to keep related features within a chapter.
1.7.1 Determining the Goals of the Text Editor

As our first step, we define what we want our text editor to do:

1. Edit a single text document.
2. Provide a menubar/menus for all functions.
3. Load from a file.
4. Save to a file.
5. Keep track of when the file has been changed.
7. Search and replace functionality.
8. Multiple views of the same text.

1.7.2 Chapter 1: A Minimal App

Let's ensure that we can set up our build process to compile and verify our code as we add features. We begin by writing a minimal program with no other purpose than opening a window.

The code for that is barely longer than a "Hello, world" program and is marked in the source code as TUTORIAL

```c
#include <FL/Fl_Double_Window.H>
#include <FL/Fl.H>

Fl_Double_Window *app_window = NULL;

void tut1_build_app_window() {
    app_window = new Fl_Double_Window(640, 480, "FLTK Editor");
}

int main (int argc, char **argv) {
    tut1_build_app_window();
    app_window->show(argc, argv);
    return Fl::run();
}
```

Passing `argc` and `argv` to `Fl_Double_Window::show()` allows FLTK to parse command line options, providing the user with the ability to change the color or graphical scheme of the editor at launch time.

`Fl::run()` will return when no more windows in the app are visible. In other words, if all windows in an app are closed, hidden, or deleted. Pressing "Escape" or clicking the "Close" button in the window frame will close our only window, prompting `Fl::run()` to return, effectively ending the app.

When building FLTK from source, the CMake environment includes the necessary rules to build the editor. You can find more information on how to write your own CMake files in the README.CMake.txt text in the top FLTK directory.

For Linux and macOS, FLTK comes with the fltk-config script that generates the compiler commands for you:

```
fltk-config --compile editor.cxx
```

If the code compiles and links correctly, running the app will pop up an empty application window on the desktop screen. You can close the window and quit the app by pressing the 'Escape' key or by clicking the "Close" button in the window frame.

Congratulations, you've just built a minimal FLTK app.
1.7 Designing a Simple Text Editor

1.7.3 Chapter 2: Adding a Menu Bar

In this chapter, we will add the main menu bar with a File menu and a Quit button. This is a good time to define a flag that will track changes in the text later.

```cpp
// remove `main()' from chapter 1, but keep the rest of the code, then add...
#include <FL/Fl_Menu_Bar.H>
#include <FL/fl_ask.H>

Fl_Menu_Bar *app_menu_bar = NULL;
bool text_changed = false;

void menu_quit_callback(Fl_Widget *, void *) { /* TODO */ }

void tut2_build_app_menu_bar() {
    app_window->begin();
    app_menu_bar = new Fl_Menu_Bar(0, 0, app_window->w(), 25);
    app_menu_bar->add("File/Quit Editor", FL_COMMAND+'q', menu_quit_callback);
    app_window->callback(menu_quit_callback);
    app_window->end();
}

int main (int argc, char **argv) {
    tut1_build_app_window();
    tut2_build_app_menu_bar();
    app_window->show(argc, argv);
    return Fl::run();
}
```

`begin()` tells FLTK to add all widgets created hereafter to our `app_window`. In this particular case, it is redundant because creating the window in the previous chapter already called `begin()` for us.

In the next line, we create the menu bar and add our first menu item to it. Menus can be constructed like file paths, with forward slashes `'/'` separating submenus from menu items.

Our basic callback is simple:

```cpp
void menu_quit_callback(Fl_Widget *, void *) {
    Fl::hide_all_windows();
}
```

`Fl::hide_all_windows()` will make all windows invisible, causing `Fl::run()` to return and `main` to exit.

The next line, `app_window->callback(menu_quit_callback)` links the same `menu_quit_callback` to the `app_window` as well. Assigning the window callback removes the default "Escape" key handling and allows the `menu_quit_callback` to handle that keypress with a friendly dialog box instead of just quitting the app.

The `Fl_Widget*` parameter in the callback will either be `app_window` if called through the window callback, or `app_menu_bar` if called by one of the menu items.

One of our goals was to keep track of text changes. If we know the text changed and is unsaved, we should notify the user that she is about to lose her work. We achieve this by adding a dialog box in the Quit callback that queries if the user really wants to quit, even if text was changed:

```cpp
void menu_quit_callback(Fl_Widget *, void *) {
    if (text_changed) {
        int c = fl_choice("Changes in your text have not been saved.\nDo you want to quit the editor anyway?", "Quit", "Cancel", NULL);
        if (c == 1) return;
        Fl::hide_all_windows();
    }
}
```
1.7.4 Chapter 3: Adding a Text Editor widget

FLTK comes with a pretty capable built-in text editing widget. We will use this `Fl_Text_Editor` widget here to allow users to edit their documents.

`Fl_Text_Editor` needs an `Fl_Text_Buffer` to do anything useful. What might seem like an unnecessary extra step is a great feature: we can assign one text buffer to multiple text editors. In a later chapter, we will use this feature to implement a split editor window.

```cpp
#include <FL/Fl_Text_Buffer.H>
#include <FL/Fl_Text_Editor.H>
#include <FL/filename.H>

Fl_Text_Editor *app_editor = NULL;
Fl_Text_Editor *app_split_editor = NULL; // for later
Fl_Text_Buffer *app_text_buffer = NULL;
char app_filename[FL_PATH_MAX] = "";

// ... callbacks go here

void tut3_build_main_editor() {
  app_window->begin();
  app_text_buffer = new Fl_Text_Buffer();
  app_text_buffer->add_modify_callback(text_changed_callback, NULL);
  app_editor = new Fl_Text_Editor(0, app_menu_bar->h(),
                                 app_window->w(), app_window->h() - app_menu_bar->h());
  app_editor->buffer(app_text_buffer);
  app_editor->textfont(FL_COURIER);
  app_window->resizable(app_editor);
  app_window->end();
}
```

By setting the `app_editor` to be the `resizable()` property of `app_window`, we make our application window resizable on the desktop, and we ensure that resizing the window will only resize the text editor vertically, but not our menu bar.

To keep track of changes to the document, we add a callback to the text editor that will be called whenever text is added or deleted. The text modify callback sets our `text_changed` flag if text was changed:

```cpp
// insert before tut3_build_main_editor()
void text_changed_callback(int, int n_inserted, int n_deleted, int, const char*, void*) {
  if (n_inserted || n_deleted)
    text_changed = true;
}
```

To wrap this chapter up, we add a "File/New" menu and link it to a callback that clears the text buffer, clears the current filename, and marks the buffer as unchanged.

```cpp
// insert before tut3_build_main_editor()
void menu_new_callback(Fl_Widget*, void*) {
  app_text_buffer->text("");
  text_changed = false;
}
```

1.7.5 Chapter 4: Reading and Writing Files

In this chapter, we will add support for loading and saving text files, so we need three more menu items in the File menu: Open, Save, and Save As.

```cpp
#include <FL/Fl_Native_File_Chooser.H>
#include <FL/platform.H>
#include <errno.h>

// ... add callbacks here

void tut4_add_file_support() {
  int ix = app_menu_bar->find_index(menu_quit_callback);
  app_menu_bar->insert(ix, "Open", FL_COMMAND+'o', menu_open_callback, NULL, FL_MENU_DIVIDER);
  app_menu_bar->insert(ix+1, "Save", FL_COMMAND+'s', menu_save_callback);
  app_menu_bar->insert(ix+2, "Save as...", FL_COMMAND+'S', menu_save_as_callback, NULL, FL_MENU_DIVIDER);
}
```
1.7 Designing a Simple Text Editor

Note

The menu shortcuts FL_COMMAND+'s' and FL_COMMAND+'S' look the same at a first glance, but the second shortcut is actually Ctrl-Shift-S due to the capital letter 'S'. Also, we use FL_COMMAND as our menu shortcut modifier key. FL_COMMAND translates to FL_CTRL on Windows and Linux, and to FL_META on macOS, better known as the cloverleaf, or simply "the Apple key".

We implement the Save As callback first, because we will want to call it from the Open callback later. The basic callback is only a few lines of code.

```c
void menu_save_as_callback(Fl_Widget*, void*) {
    Fl_Native_File_Chooser file_chooser;
    file_chooser.title("Save File As...");
    file_chooser.type(Fl_Native_File_Chooser::BROWSE_SAVE_FILE);
    if (file_chooser.show() == 0) {
        app_text_buffer->savefile(file_chooser.filename());
        strncpy(app_filename, file_chooser.filename(), FL_PATH_MAX-1);
        text_changed = false;
    }
}
```

However if the user has already set a file name including path information, it is the polite thing to preload the file chooser with that information. This little chunk of code will separate the file name from the path before we call file_chooser.show():

```c
// insert before 'if (file_chooser.show()...'
if (app_filename[0]) {
    char temp_filename[FL_PATH_MAX];
    strncpy(temp_filename, app_filename, FL_PATH_MAX-1);
    const char *name = fl_filename_name(temp_filename);
    if (name) {
        file_chooser.preset_file(name);
        temp_filename[name - temp_filename] = 0;
        file_chooser.directory(temp_filename);
    }
}
```

Great. Now let's add code for our File/Save menu. If no filename was set yet, it falls back to our Save As callback. FL_Text_Editor::savefile() writes the contents of our text widget into a UTF-8 encoded text file.

```c
void menu_save_callback(Fl_Widget*, void*) {
    if (!app_filename[0]) {
        menu_save_as_callback(NULL, NULL);
    } else {
        app_text_buffer->savefile(file_chooser.filename());
        text_changed = false;
    }
}
```

On to loading a new file. Let's write the function to load a file from a given file name:

```c
void load(const char *filename) {
    if (app_text_buffer->loadfile(filename) == 0) {
        strncpy(app_filename, filename, FL_PATH_MAX-1);
        text_changed = false;
    } else {
        fl_alert("Failed to load file
n%s
n%s", filename, strerror(erno));
    }
}
```

A friendly app should warn the user if file operations fail. This can be done in three lines of code, so let's add an alert dialog after every loadfile and savefile call. This is exemplary for load(), and the code is very similar for the two other locations.

```c
void load(const char *filename) {
    if (app_text_buffer->loadfile(filename) == 0) {
        strncpy(app_filename, filename, FL_PATH_MAX-1);
        text_changed = false;
    } else {
        fl_alert("Failed to load file\n%s\n%s", filename, strerror(erno));
    }
}
```

If the user selects our pulldown "Load" menu, we first check if the current text was modified and provide a dialog box that offers to save the changes before loading a new text file:

```c
void menu_open_callback(Fl_Widget*, void*) {
    if (text_changed) {
        int r = fl_choice("The current file has not been saved.\n" "Would you like to save it now?", "Cancel", "Save", "Don't Save");
        if (r == 2)
```
If the user did not cancel the operation, we pop up a file chooser for loading the file, using similar code as in Save As.

```c
...  
    Fl_Native_File_Chooser file_chooser;  
    file_chooser.title("Open File...");  
    file_chooser.type(Fl_Native_File_Chooser::BROWSE_FILE);  
...  

Again, we preload the file chooser with the last used path and file name:

```c
...  
    if (app_filename[0]) {  
        char temp_filename[FL_PATH_MAX];  
        strncpy(temp_filename, app_filename, FL_PATH_MAX-1);  
        const char *name = fl_filename_name(temp_filename);  
        if (name) {  
            file_chooser.preset_file(name);  
            temp_filename[name - temp_filename] = 0;  
            file_chooser.directory(temp_filename);  
        }  
    }  
...  
```

And finally, we pop up the file chooser. If the user cancels the file dialog, we do nothing and keep the current file. Otherwise, we call the `load()` function that we already wrote:

```c
...  
    if (file_chooser.show() == 0)  
        load(file_chooser.filename());  
...  
```

We really should support two more ways to load documents from a file. Let's modify the "show and run" part of `main()` to handle command line parameters and desktop drag'n'drop operations. For that, we refactor the last two lines of `main()` into a new function:

```c
// ... new function here
int main (int argc, char **argv) {  
    tut1_build_app_window();  
    tut2_build_app_menu_bar();  
    tut3_build_main_editor();  
    tut4_add_file_support();  
    // ... refactor those into the new function  
    // app_window->show(argc, argv);  
    // return Fl::run();  
    return tut4_handle_commandline_and_run(argc, argv);  
}  
```

Our function to show the window and run the app has a few lines of boilerplate code. `Fl::args_to_utf8()` converts the command line argument from whatever the host system provides into Unicode. `Fl::args()` goes through the list of arguments and gives `args_handler()` a chance to handle each argument. It also makes sure that FLTK specific args are still forwarded to FLTK, so "-scheme plastic" and "-background #aaccff" will draw beautiful blue buttons in a plastic look.

`fl_open_callback()` lets FLTK know what to do if a user drops a text file onto our editor icon (Apple macOS). Here, we ask it to call the `load()` function that we wrote earlier.

```c
// ... args_handler here
int tut4_handle_commandline_and_run(int argc, char **argv) {  
    int i = 0;  
    Fl::args_to_utf8(argc, argv);  
    Fl::args(argc, argv, i, args_handler);  
    fl_open_callback(load);  
    app_window->show(argc, argv);  
    return Fl::run();  
}  
```

Last work item for this long chapter: what should our `args_handler` do? We could handle additional command line options here, but for now, all we want to handle is file names and paths. Let's make this easy: if the current arg does not start with a '-', we assume it is a file name, and we call `load()`:

```c
int args_handler(int argc, char **argv, int &i) {  
    if (argv[i++][0] != '-') {  
        load(argv[i]);  
    }  
}  
```
1.7 Designing a Simple Text Editor

```c
i++;     
return 1;
}
return 0;
```

So this is our basic but quite functional text editor app in about 100 lines of code. The following chapters add some user convenience functions and show off some FLTK features including split editors and syntax highlighting.

1.7.6 Chapter 5: Cut, Copy, and Paste

The FLTK Text Editor widget comes with built-in cut, copy, and paste functionality, but as a courtesy, we should also offer these as menu items in the main menu.

In our feature list, we noted that we want to implement a split text editor. This requires that the callbacks know which text editor has the keyboard focus. Calling `Fl::focus()` may return `NULL` or other unknown widgets, so we add a little test in our callbacks:

```c
void menu_cut_callback(Fl_Widget*, void* v) {
  Fl_Widget *e = Fl::focus();
  if (e && (e == app_editor || e == app_split_editor))
    Fl_Text_Editor::kf_cut(0, (Fl_Text_Editor*)e);
}
```

We can write very similar callbacks for undo, redo, copy, paste, and delete. Adding a new menu and the six menu items follows the same pattern as before. Using the Menu/Item notation will create an Edit menu for us:

```c
void tut5_cut_copy_paste() {
  app_menu_bar->add("Edit/Undo", FL_COMMAND+'z', menu_undo_callback);
  app_menu_bar->add("Edit/Redo", FL_COMMAND+'Z', menu_redo_callback, NULL, FL_MENU_DIVIDER);
  app_menu_bar->add("Edit/Cut", FL_COMMAND+'x', menu_cut_callback);
  app_menu_bar->add("Edit/Copy", FL_COMMAND+'c', menu_copy_callback);
  app_menu_bar->add("Edit/Paste", FL_COMMAND+'v', menu_paste_callback);
  app_menu_bar->add("Edit/Delete", 0, menu_delete_callback);
}
```

1.7.7 Chapter 6: Find and Find Next

Corporate called. They want a dialog box for their users that can search for some word in the text file. We can add this functionality using a callback and a standard FLTK dialog box.

Here is some code to find a string in a text editor. The first four lines make sure that we start our search at the cursor position of the current editor window. The rest of the code searches the string and marks it if found.

```c
void find_next(const char *needle) {
  Fl_Text_Editor *editor = app_editor;
  Fl_Widget *e = Fl::focus();
  if (e && (e == app_editor || e == app_split_editor))
    editor = app_split_editor;
  int pos = editor->insert_position();
  int found = app_text_buffer->search_forward(pos, needle, &pos);
  if (found) {
    app_text_buffer->select(pos, pos + (int)strlen(needle));
    editor->insert_position(pos + (int)strlen(needle));
    editor->show_insert_position();
  } else {
    fl_alert("No further occurrences of '%s' found!", needle);
  }
}
```

The callbacks are short, using the FLTK text field dialog box and the `find_next` function that we already implemented. The last searched text is saved in `last_find_text` to be reused by `menu_find_next_callback`. If no search text was set yet, or it was set to an empty text, "Find Next" will forward to `menu_find_callback` and pop up our "Find Text" dialog.

```c
char last_find_text[1024] = "";
void menu_find_callback(Fl_Widget*, void* v) {
  const char *find_text = fl_input("Find in text:", last_find_text);
  if (find_text) {
    strncpy(last_find_text, find_text, sizeof(last_find_text)-1);
    find_next(find_text);
  }
}
```

Generated by Doxygen
void menu_find_next_callback(Fl_Widget*, void* v) {
    if (last_find_text[0]) {
        find_next(last_find_text);
    } else {
        menu_find_callback(NULL, NULL);
    }
}

And of course we need to add two menu items to our main application menu.
...
app_menu_bar->add("Find/Find...", FL_COMMAND+'f', menu_find_callback);
app_menu_bar->add("Find/Find Next", FL_COMMAND+'g', menu_find_next_callback, NULL, FL_MENU_DIVIDER);
...

1.7.8 Chapter 7: Replace and Replace Next

To implement the next feature, we will need to implement our own “Find and Replace” dialog box. To make this dialog box useful, it needs the following elements:

• a text input field for the text that we want to find
• a text input field for the replacement text
• a button to find the next occurrence
• a button to replace the current text and find the next occurrence
• a button to close the dialog

This is rather complex functionality, so instead of adding more global variables, we will pack this dialog into a class, derived from Fl_Window.

Let's implement the text replacement code first:
char last_replace_text[1024] = ""

void replace_selection(const char *new_text) {
    Fl_Text_Editor *editor = app_editor;
    Fl_Widget *e = Fl::focus();
    if (e && e == app_split_editor)
        editor = app_split_editor;
    int start, end;
    if (app_text_buffer->selection_position(&start, &end)) {
        app_text_buffer->remove_selection();
        app_text_buffer->insert(start, new_text);
        app_text_buffer->select(start, start + (int)strlen(new_text));
        editor->insert_position(start + (int)strlen(new_text));
        editor->show_insert_position();
    }
}

As before, the first four lines anticipate a split editor and find the editor that has focus. The code then deletes the currently selected text, replaces it with the new text, selects the new text, and finally sets the text cursor to the end of the new text.
The Replace_Dialog class

The Replace_Dialog class holds pointers to our active UI elements as well as all the callbacks for the dialog buttons.

```cpp
class Replace_Dialog : public Fl_Double_Window {
  Fl_Input *find_text_input;
  Fl_Input *replace_text_input;
  Fl_Button *find_next_button;
  Fl_Button *replace_and_find_button;
  Fl_Button *close_button;

public:
  Replace_Dialog(const char *label);
  void show() FL_OVERRIDE;
private:
  static void find_next_callback(Fl_Widget*, void*);
  static void replace_and_find_callback(Fl_Widget*, void*);
  static void close_callback(Fl_Widget*, void*);
};

Replace_Dialog *replace_dialog = NULL;
```

The constructor creates the dialog and marks it as "non modal". This will make the dialog hover over the application window like a toolbox window until the user closes it, allowing multiple "find and replace" operations. So here is our constructor:

```cpp
Replace_Dialog::Replace_Dialog(const char *label)
  : Fl_Double_Window(430, 110, label)
  {
    find_text_input = new Fl_Input(100, 10, 320, 25, "Find:oug");
    replace_text_input = new Fl_Input(100, 40, 320, 25, "Replace:oug");
    Fl_Flex* button_field = new Fl_Flex(100, 70, w()-100, 40);  
    button_field->type(Fl_Flex::HORIZONTAL);
    button_field->margin(0, 5, 10, 10);
    button_field->gap(10);
    find_next_button = new Fl_Button(0, 0, 0, 0, "Next");
    find_next_button->callback(find_next_callback, this);
    replace_and_find_button = new Fl_Button(0, 0, 0, 0, "Replace");
    replace_and_find_button->callback(replace_and_find_callback, this);
    close_button = new Fl_Button(0, 0, 0, 0, "Close");
    close_button->callback(close_callback, this);
    button_field->end();
    set_non_modal();
  }
```

All buttons are created inside an `Fl_Flex` group. They will be arranged automatically by `Fl_Flex`, so there is no need to set x and y coordinates or a width or height. `button_field` will lay out the buttons for us.

Note

There is no need to write a destructor or delete individual widgets. When we delete an instance of `Replace_Dialog`, all children are deleted for us.

The `show()` method overrides the window's show method. It adds some code to preload the values of the text fields for added convenience. It then pops up the dialog box by calling the original `Fl_Double_Window::show()`.

```cpp
void Replace_Dialog::show() {
  find_text_input->value(last_find_text);
  replace_text_input->value(last_replace_text);
  Fl_Double_Window::show();
}
```

The buttons in the dialog need callbacks to be useful. If callbacks are defined within a class, they must be defined static, but a pointer to the class can be provided through the `user_data` field. We have done that in the constructor by adding `this` as the last argument when setting the callback, for example in `close_button->callback(close_callback, close_callback, this);`.

The callback itself can then extract the `this` pointer with a static cast:

```cpp
void Replace_Dialog::close_callback(Fl_Widget*, void* my_dialog) {
  Replace_Dialog *dlg = static_cast<Replace_Dialog*>(my_dialog);
  dlg->hide();
}
```

The callback for the Find button uses our already implemented `find_next` function:

```cpp
void Replace_Dialog::find_next_callback(Fl_Widget*, void* my_dialog) {
  Fl_Double_Window *dlg = static_cast<Replace_Dialog*>(my_dialog);
  strncpy(last_find_text,dlg->find_text_input->value(), sizeof(last_find_text)-1);
```
strncpy(last_replace_text, dlg->replace_text_input->value(), sizeof(last_replace_text)-1);
if (last_find_text[0])
    find_next(last_find_text);
}

The Replace button callback calls our newly implemented replace_selection function and then continues on to the find_next_callback:
void Replace_Dialog::replace_and_find_callback(Fl_Widget*, void* my_dialog) {
    Replace_Dialog *dlg = static_cast<Replace_Dialog*>(my_dialog);
    replace_selection(dlg->replace_text_input->value());
    find_next_callback(NULL, my_dialog);
}

This long chapter comes close to its end. We are missing menu items that pop up our dialog and that allow a quick "Replace and Find Next" functionality without popping up the dialog. The code is quite similar to the "Find" and "Find Next" code in the previous chapter:
void menu_replace_callback(Fl_Widget*, void*) {
    if (!replace_dialog)
        replace_dialog = new Replace_Dialog("Find and Replace");
    replace_dialog->show();
}

void menu_replace_next_callback(Fl_Widget*, void*) {
    if (!last_find_text[0]) {
        menu_replace_callback(NULL, NULL);
    } else {
        replace_selection(last_replace_text);
        find_next(last_find_text);
    }
}

void tut7_implement_replace() {
    app_menu_bar->add("Find/Replace...", FL_COMMAND+'r', menu_replace_callback);
    app_menu_bar->add("Find/Replace Next", FL_COMMAND+'t', menu_replace_next_callback);
}

1.7.9 Chapter 8: Editor Features

Chapter 7 was long an intense. Let’s relax and implement something simple here. We want menus with check boxes that can toggle some text editor features on and off:
void tut8_editor_features() {
    app_menu_bar->add("Window/Line Numbers", FL_COMMAND+'l', menu_linenumbers_callback, NULL, FL_MENU_TOGGLE);
    app_menu_bar->add("Window/Word Wrap", 0, menu_wordwrap_callback, NULL, FL_MENU_TOGGLE);
}

The Fl_Widget parameter in callbacks always points to the widget that causes the callback. Menu items are not derived from widgets, so to find out which menu item caused a callback, we can do this:
void menu_linenumbers_callback(Fl_Widget*, void*) {
    Fl_Menu_Bar* menu = static_cast<Fl_Menu_Bar*>(w);
    const Fl_Menu_Item* linenumber_item = menu->mvalue();
    if (linenumber_item->value()) {
        app_editor->linenumber_width(40);
    } else {
        app_editor->linenumber_width(0);
    }
    app_editor->redraw();
}

Setting the width enables the line numbers, setting it to 0 disables the line number display. When changing the value of a widget, FLTK will make sure that the widget is redrawn to reflect the new value. When changing other attributes such as colors or fonts, FLTK assumes that many attributes are changed at the same time and leaves it to the user to call Fl_Widget::redraw() when done. Here we call app_editor->redraw() to make sure that the change in the line number setting is also drawn on screen.

Let’s not forget to update the line number display for a potential split editor widget as well:
// add before the end of menu_linenumbers_callback
if (app_split_editor) {
    const Fl_Menu_Item* linenumber_item = menu->mvalue();
    if (linenumber_item->value()) {
        app_split_editor->linenumber_width(40);
    } else {
        app_split_editor->linenumber_width(0);
    }
    app_split_editor->redraw();
}

The word wrap feature is activated by calling Fl_Text_Editor::wrap_mode() with the parameters Fl_Text_Display::WRAP_AT_BOUNDS and 0. It's deactivated with Fl_Text_Display::WRAP_NONE. The implementation of the callback is the same as menu_linenumbers_callback.
1.7 Designing a Simple Text Editor

1.7.10 Chapter 9: Split Editor

When editing long source code files, it can be really helpful to split the editor to view statements at the top of the text while adding features at the bottom of the text in a split text view.

FLTK can link multiple text editors to a single text buffer. Let’s implement this now. This chapter will show you how to rearrange widgets in an existing widget tree.

Our initializer removes the main text editor from the widget tree and replaces it with an Fl_Tile. A tile can hold multiple widgets that can then be resized interactively by the user by clicking and dragging the divider between those widgets.

We start by replacing the editor widget with a tile group of the same size.

```c
#include <FL/Fl_Tile.H>
Fl_Tile *app_tile = NULL;

void tut9_split_editor() {
    app_window->begin();
    app_tile = new Fl_Tile(app_editor->x(), app_editor->y(),
                           app_editor->w(), app_editor->h());
    app_window->remove(app_editor);

    Next we add our existing editor as the first child of the tile and create another text editor app_split_editor as the second child of the tile, but it’s hidden for now with a height of zero pixels.

    Note
    Creating the new Fl_Tile also calls Fl_Tile::begin().

    Adding app_editor to the tile would have also removed it from app_window, so app_window->remove(app_editor) in the code above is not really needed, but illustrates what we are doing.
    app_tile->add(app_editor);
    app_split_editor = new Fl_Text_Editor(app_tile->x(), app_tile->y()+app_tile->h(),
                                          app_tile->w(), 0);
    app_split_editor->buffer(app_text_buffer);
    app_split_editor->textfont(FL_COURIER);
    app_split_editor->hide();

    Now we clean up after ourselves and make sure that the resizables are all set correctly. Lastly, we add a menu item with a callback.
    app_tile->end();
    app_tile->size_range(0, 25, 25);
    app_tile->size_range(1, 25, 25);
    app_window->end();
    app_window->resizable(app_tile);
    app_tile->resizable(app_editor);
    app_menu_bar->add("Window/Split", FL_COMMAND+'-', menu_split_callback, NULL, FL_MENU_TOGGLE);
}
```

Now with all widgets in place, the callback’s job is to show and resize, or hide and resize the split editor. We can implement that like here:

```c
void menu_split_callback(Fl_Widget* w, void*) {
    Fl_Menu_Bar* menu = static_cast<Fl_Menu_Bar*>(w);
    const Fl_Menu_Item* splitview_item = menu->mvalue();
    if (splitview_item->value()) {
        int h_split = app_tile->h()/2;
        app_editor->size(app_tile->w(), h_split);
        app_split_editor->resize(app_tile->x(), app_tile->y() + h_split,
                                  app_tile->w(), app_tile->h() - h_split);
        app_split_editor->show();
    } else {
        app_editor->size(app_tile->w(), app_tile->h());
        app_split_editor->resize(app_tile->x(), app_tile->y()+app_tile->h(),
                                 app_tile->w(), 0);
        app_split_editor->hide();
    }
    app_tile->resizable(app_editor);
    app_tile->init_sizes();
    app_tile->redraw();
}
```

Generated by Doxygen
1.7.11 Chapter 10: Syntax Highlighting

Chapter 10 adds a lot of code to implement "C" language syntax highlighting. Not all code is duplicated here in the documentation. Please check out test/editor.cxx for all the details.

The Fl_Text_Editor widget supports highlighting of text with different fonts, colors, and sizes. The implementation is based on the excellent NEdit text editor core, from https://sourceforge.net/projects/nedit/, which uses a parallel "style" buffer which tracks the font, color, and size of the text that is drawn.

Styles are defined using the Fl_Text_Display::Style_Table_Entry structure defined in <FL/Fl_Text_Display.H>:

```c
struct Style_Table_Entry {
  FL_Color color;
  FL_Font font;
  int size;
  unsigned attr;
};
```

The `color` member sets the color for the text, the `font` member sets the FLTK font index to use, and the `size` member sets the pixel size of the text. The `attr` member is currently not used.

For our text editor we'll define 7 styles for plain code, comments, keywords, and preprocessor directives:

```c
Fl_Text_Display::Style_Table_Entry styletable[] = {
  { FL_BLACK, FL_COURIER, FL_NORMAL_SIZE }, // A - Plain
  { FL_DARK_GREEN, FL_COURIER_ITALIC, FL_NORMAL_SIZE }, // B - Line comments
  { FL_DARK_GREEN, FL_COURIER_ITALIC, FL_NORMAL_SIZE }, // C - Block comments
  { FL_BLUE, FL_COURIER, FL_NORMAL_SIZE }, // D - Strings
  { FL_DARK_RED, FL_COURIER, FL_NORMAL_SIZE }, // E - Directives
  { FL_DARK_RED, FL_COURIER_BOLD, FL_NORMAL_SIZE }, // F - Types
  { FL_BLUE, FL_COURIER_BOLD, FL_NORMAL_SIZE } // G - Keywords
};
```

You'll notice that the comments show a letter next to each style - each style in the style buffer is referenced using a character starting with the letter 'A'.

You call the `highlight_data()` method to associate the style data and buffer with the text editor widget:

```c
app_editor->highlight_data(app_style_buffer, styletable, sizeof(styletable) / sizeof(styletable[0]), 'A', style_unfinished_cb, 0);
```

Finally, you need to add a callback to the main text buffer so that changes to the text buffer are mirrored in the style buffer:

```c
app_text_buffer->add_modify_callback(style_update, app_editor);
```

The `style_update()` function, like the `change_cb()` function described earlier, is called whenever text is added or removed from the text buffer. It mirrors the changes in the style buffer and then updates the style data as necessary:

```c
// 'style_update()' - Update the style buffer...

void style_update(int pos, // I - Position of update
                  int nInserted, // I - Number of inserted chars
                  int nDeleted, // I - Number of deleted chars
                  int nRestyled, // I - Number of restyled chars
                  const char *deletedText, // I - Text that was deleted
                  void (*cbArg) { // I - Callback data
  int start, // Start of text
       end; // End of text
  char last; // Last style on line
  // Style data
  *text;
}
```

// If this is just a selection change, just unselect the style buffer...
if (nInserted == 0 && nDeleted == 0) {
  app_style_buffer->unselect();
  return;
```
1.7 Designing a Simple Text Editor

```c
if (nInserted > 0) {
  // Insert characters into the style buffer...
  style = new char[nInserted + 1];
  memset(style, 'A', nInserted);
  style[nInserted] = '\0';
  app_style_buffer->replace(pos, pos + nDeleted, style);
  delete[] style;
} else {
  // Just delete characters in the style buffer...
  app_style_buffer->remove(pos, pos + nDeleted);
}
```

The `style_parse()` function scans a copy of the text in the buffer and generates the necessary style characters for display. It assumes that parsing begins at the start of a line:

```c
for (current = *style, col = 0, last = 0; length > 0; text ++) {
  // Check for directives, comments, strings, and keywords...
  if (current == '\*') {
    // Set style to directive current = 'E';
    } else if (strncmp(text, "/*", 2) == 0) {
    current = 'C';
    } else if (strncmp(text, "\\\\", 2) == 0) {
    // Quoted quote...
    *style++ = current;
```
*style++ = current;
text ++;
length --;
col += 2;
continue;
} else if (*text == '\"') {
current = 'D';
if (!last && islower(*text)) {
  // Might be a keyword...
  for (temp = text, bufptr = buf;
     islower(*temp) && bufptr < (buf + sizeof(buf) - 1);
     bufptr++ = *temp++);
  if (*last && islower(*temp)) {
    *bufptr = '\"';
    bufptr = buf;
  }
  if (bsearch(bufptr, code_types,
      sizeof(code_types) / sizeof(code_types[0]),
      sizeof(code_types[0]), compare_keywords)) {
    while (text < temp) {
      *style++ = 'F';
text ++;
      length --;
col ++;
    }
text --;
    length ++;
    last = 1;
    continue;
  } else if (bsearch(bufptr, code_keywords,
      sizeof(code_keywords) / sizeof(code_keywords[0]),
      sizeof(code_keywords[0]), compare_keywords)) {
    while (text < temp) {
      *style++ = 'G';
text ++;
      length --;
col ++;
    }
text --;
    length ++;
    last = 1;
    continue;
  }
}
} else if (*text == '\n') {
  // Close a C comment...
  *style++ = current;
  *style++ = current;
text ++;
  length --;
col += 2;
continue;
} else if (*text == '\"') {
  // Quoted end quote...
  *style++ = current;
  *style++ = current;
text ++;
  length --;
col += 2;
continue;
} else if (*text == '\"') {
  // End quote...
  *style++ = current;
col ++;
current = 'A';
continue;
}
}
}
} else if (*text == ('\' || *text == '\')) {
  // Copy style info...
  if (current == 'A' && strncmp(text, "*/", 2) == 0) {
    // Close a C comment...
    *style++ = current;
    *style++ = current;
text ++;
    length --;
col += 2;
    continue;
  } else if (strncmp(text, "\\", 2) == 0) {
    // Quoted end quote...
    *style++ = current;
    *style++ = current;
text ++;
    length --;
col += 2;
    continue;
  } else if (*text == '\"') {
    // End quote...
    *style++ = current;
col ++;
current = 'A';
    continue;
  }
}
if (current == 'A' && (*text == '{' || *text == '}')) {
  *style++ = 'G';
else {
  *style++ = current;
col ++;
}
last = isalnum(*text) || *text == '.';
if (*text == '\n') {
  // Reset column and possibly reset the style
  col = 0;
}
1.8 Drawing Things in FLTK

This chapter covers the drawing functions that are provided with FLTK.

1.8.1 When Can You Draw Things in FLTK?

There are only certain places you can execute FLTK code that draws to the computer's display. Calling these functions at other places will result in undefined behavior!

- The most common place is inside the virtual `Fl_Widget::draw()` method. To write code here, you must subclass one of the existing `Fl_Widget` classes and implement your own version of `draw()`.

- You can also create custom `boxtypes` and `labeltypes`. These involve writing small procedures that can be called by existing `Fl_Widget::draw()` methods. These "types" are identified by an 8-bit index that is stored in the widget's `box()`, `labeltype()`, and possibly other properties.

- You can call `Fl_Window::make_current()` to do incremental update of a widget. Use `Fl_Widget::window()` to find the window.

In contrast, code that draws to other drawing surfaces than the display (i.e., instances of derived classes of the `Fl_Surface_Device` class, except `Fl_Display_Device`, such as `Fl_Printer` and `Fl_Copy_Surface`) can be executed at any time as follows:

1. Make your surface the new current drawing surface calling the `Fl_Surface_Device::push_current(Fl_Surface_Device *)` function.
2. Make a series of calls to any of the drawing functions described below; these will operate on the new current drawing surface;
3. Set the current drawing surface back to its previous state calling `Fl_Surface_Device::pop_current()`.

1.8.2 What Units Do FLTK Functions Use?

Before version 1.4 all graphical quantities used by FLTK were in pixel units: a window of width 500 units was 500 pixels wide, a line of length 10 units was 10 pixels long, lines of text written using a 14-point font were 14 pixels below each other. This organization is not sufficient to support GUI apps that can be drawn on screens of varying pixel density, especially on High-DPI screens, because widgets become very small and text becomes unreadable.

FLTK version 1.4 introduces a new feature, a screen-specific `scale factor` which is a float number with a typical value in the 1-2.5 range and is used as follows: any graphical element with an FLTK value of \( v \) units is drawn on the screen with \( v \times \text{scale} \) units. Thus, a window with width 500 units is 500\times scale pixels wide, a line of length 10 units is 10\times scale pixels long, lines of text written using a 14-point font are 14\times scale pixels below each other. Consider a system with two screens, one with regular DPI and one with a twice higher DPI. If the first screen's scale factor is set to 1 and that of the second screen to 2, the GUI of any FLTK app appears equally sized on the two screens.

FLTK uses several units to measure graphical elements:
• All quantities used by the public FLTK API to measure graphical elements (e.g., window widths, line lengths, font sizes, clipping regions, image widths and heights) are in FLTK units except if it’s explicitly documented another unit is used. FLTK units are both platform- and DPI-independent. An example of FLTK API using another unit is Fl_Gl_Window::pixel_w().

• Just before drawing to a screen, the library internally multiplies all quantities expressed in FLTK units by the current value of the scale factor for the screen in use and obtains quantities in drawing units. The current scale factor value, for an Fl_Window named window, is given by

```cpp
int nscreen = window->screen_num(); // the screen where window is mapped
float s = Fl::screen_scale(nscreen); // this screen’s scale factor
```

One drawing unit generally corresponds to one screen pixel ...

• ... but not on macOS and for retina displays, where one drawing unit corresponds to two pixels.

• ... and not with the Wayland platform, where one drawing unit may correspond to 1, 2, or 3 pixels according to the current value of the Wayland-defined, integer-valued scale factor.

At application start time, FLTK attempts to detect the adequate scale factor value for each screen of the system. Here is how that’s done under the X11, Windows, and Wayland platforms. If the resulting scale factor is not satisfactory, and also under the macOS platform, it's possible to set the FLTK_SCALING_FACTOR environmental variable to the desired numerical value (e.g., 1.75) and any FLTK app will start scaled with that value. Furthermore, it's possible to change the scale factor value of any screen at run time with ctrl/+/-/0/ keystrokes which enlarge, shrink, and reset, respectively, all FLTK windows on a screen and their content. Under macOS, the corresponding GUI scaling shortcuts are cmd/+/-/0/.

GUI rescaling involves also image drawing: the screen area covered by the drawn image contains a number of pixels that grows with the scale factor. When FLTK draws images, it maps the image data (the size of these data is given by Fl_Image::data_w() and Fl_Image::data_h()) to the screen area whose size (in FLTK units) is given by Fl_Image::w() and Fl_Image::h()). How exactly such mapping is performed depends on the image type, the platform and some hardware features. The most common case for Fl_RGB_Images is that FLTK uses a scaled drawing system feature that directly maps image data to screen pixels. An important feature of FLTK for image drawing is the Fl_Image::scale() member function, new in FLTK version 1.4. This function controls the image drawing size (in FLTK units, given by Fl_Image::w() and Fl_Image::h()) independently from the size of the image data (given by Fl_Image::data_w() and Fl_Image::data_h()). An image with large enough data size can thus be drawn at the full resolution of the screen even when the screen area covered by the image grows following the GUI scale factor.

The Fl_Image_Surface class is intended to create an Fl_RGB_Image from a series of FLTK drawing operations. The Fl_Image_Surface constructor allows to control whether the size in pixels of the resulting image matches the FLTK units used when performing drawing operations, or matches the number of pixels corresponding to these FLTK units given the current value of the scale factor. The first result is obtained with new Fl_Image_Surface(w, h), the second with new Fl_Image_Surface(w, h, 1).

When drawing to Fl_Printer or Fl_PostScript_File_Device, the drawing unit is initially one point, that is, 1/72 of an inch. This unit is changed by calls to Fl_Paged_Device::scale().

### 1.8.3 Drawing Functions

To use the drawing functions you must first include the `<FL/fl_draw.H>` header file. FLTK provides the following types of drawing functions:

- Boxes
- Clipping
- Colors
- Line Dashes and Thickness
1.8 Drawing Things in FLTK

- Drawing Fast Shapes
- Drawing Complex Shapes
- Drawing Text
- Fonts
- Character Encoding
- Drawing Overlays
- Drawing Images
- Direct Image Drawing
- Direct Image Reading
- Image Classes
- Offscreen Drawing

1.8.3.1 Boxes

FLTK provides three functions that can be used to draw boxes for buttons and other UI controls. Each function uses the supplied upper-lefthand corner and width and height to determine where to draw the box.

```cpp
void fl_draw_box(Fl_Boxtype b, int x, int y, int w, int h, Fl_Color c)
```

The `fl_draw_box()` function draws a standard boxtypet b in the specified color c.

```cpp
void fl_frame(const char *s, int x, int y, int w, int h)
void fl_frame2(const char *s, int x, int y, int w, int h)
```

The `fl_frame()` and `fl_frame2()` functions draw a series of line segments around the given box. The string s must contain groups of 4 letters which specify one of 24 standard grayscale values, where ‘A’ is black and ‘X’ is white. The results of calling these functions with a string that is not a multiple of 4 characters in length are undefined.

The only difference between `fl_frame()` and `fl_frame2()` is the order of the line segments:

- For `fl_frame()` the order of each set of 4 characters is: top, left, bottom, right.
- For `fl_frame2()` the order of each set of 4 characters is: bottom, right, top, left.

Note that `fl_frame(Fl_Boxtype b)` is described in the Box Types section.
1.8.3.2 Clipping

You can limit all your drawing to a rectangular region by calling `fl_push_clip()`, and put the drawings back by using `fl_pop_clip()`. This rectangle is measured in FLTK units and is unaffected by the current transformation matrix.

In addition, the system may provide clipping when updating windows which may be more complex than a simple rectangle.

```c
void fl_push_clip(int x, int y, int w, int h)
void fl_clip(int x, int y, int w, int h)
```

Intersect the current clip region with a rectangle and push this new region onto the stack.

The `fl_clip()` version is deprecated and will be removed from future releases.

```c
void fl_push_no_clip()
```

Pushes an empty clip region on the stack so nothing will be clipped.

```c
void fl_pop_clip()
```

Restore the previous clip region.

**Note:** You must call `fl_pop_clip()` once for every time you call `fl_push_clip()`. If you return to FLTK with the clip stack not empty unpredictable results occur.

```c
int fl_not_clipped(int x, int y, int w, int h)
```

Returns non-zero if any of the rectangle intersects the current clip region. If this returns 0 you don't have to draw the object.

**Note:** Under X this returns 2 if the rectangle is partially clipped, and 1 if it is entirely inside the clip region.

```c
int fl_clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H)
```

Intersect the rectangle \(x, y, w, h\) with the current clip region and returns the bounding box of the result in \(X, Y, W, H\). Returns non-zero if the resulting rectangle is different than the original. This can be used to limit the necessary drawing to a rectangle. \(W\) and \(H\) are set to zero if the rectangle is completely outside the region.

```c
void fl_clip_region(Fl_Region r)
Fl_Region fl_clip_region()
```

Replace the top of the clip stack with a clipping region of any shape. `Fl_Region` is an operating system specific type. The second form returns the current clipping region.
1.8.3.3 Colors

FLTK manages colors as 32-bit unsigned integers, encoded as RGBI. When the "RGB" bytes are non-zero, the value is treated as RGB. If these bytes are zero, the "I" byte will be used as an index into the colormap. Colors with both "RGB" set and an "I" > 0 are reserved for special use.

Values from 0 to 255, i.e. the "I" index value, represent colors from the FLTK standard colormap and are allocated as needed on screens without TrueColor support. The `Fl_Color` enumeration type defines the standard colors and color cube for the first 256 colors. All of these are named with symbols in `<FL/Enumerations.H>`. Example:

![FLTK default colormap](image)

Color values greater than 255 are treated as 24-bit RGB values. These are mapped to the closest color supported by the screen, either from one of the 256 colors in the FLTK colormap or a direct RGB value on TrueColor screens.

```c
Fl_Color fl_rgb_color(uchar r, uchar g, uchar b)
Fl_Color fl_rgb_color(uchar grayscale)
```

Generate `Fl_Color` out of specified 8-bit RGB values or one 8-bit grayscale value.

```c
void fl_color(Fl_Color c)
void fl_color(int c)
```

Sets the color for all subsequent drawing operations. Please use the first form: the second form is only provided for back compatibility.
For colormapped displays, a color cell will be allocated out of \texttt{fl_colormap} the first time you use a color. If the colormap fills up then a least-squares algorithm is used to find the closest color.

\texttt{Fl\_Color fl\_color()}

Returns the last color that was set using \texttt{fl\_color()}. This can be used for state save/restore.

\texttt{void fl\_color(uchar r, uchar g, uchar b)}

Set the color for all subsequent drawing operations. The closest possible match to the RGB color is used. The RGB color is used directly on TrueColor displays. For colormap visuals the nearest index in the gray ramp or color cube is used.

\texttt{unsigned Fl::get\_color(Fl\_Color i)}

\texttt{void Fl::get\_color(Fl\_Color i, uchar &red, uchar &green, uchar &blue)}

Generate RGB values from a colormap index value \texttt{i}. The first returns the RGB as a 32-bit unsigned integer, and the second decomposes the RGB into three 8-bit values.

\texttt{Fl::get\_system\_colors()}

\texttt{Fl::foreground()}

\texttt{Fl::background()}

\texttt{Fl::background2()}

The first gets color values from the user preferences or the system, and the other routines are used to apply those values.

\texttt{Fl::own\_colormap()}

\texttt{Fl::free\_color(Fl\_Color i, int overlay)}

\texttt{Fl::set\_color(Fl\_Color i, unsigned c)}

\texttt{Fl::own\_colormap()} is used to install a local colormap [X11 only].

\texttt{Fl::free\_color()} and \texttt{Fl::set\_color()} are used to remove and replace entries from the colormap.

There are two predefined graphical interfaces for choosing colors. The function \texttt{fl\_show\_colormap()} shows a table of colors and returns an \texttt{Fl\_Color} index value. The \texttt{Fl\_Color\_Chooser} widget provides a standard RGB color chooser.

As the \texttt{Fl\_Color} encoding maps to a 32-bit unsigned integer representing RGBI, it is also possible to specify a color using a hex constant as a color map index:
or specify a color using a hex constant for the RGB components:

```cpp
// RGB COLOR ASSIGNMENTS
color(0xRRGGBB00)
  | | | Must be zero
  | | Blue (8 bits)
  | Green (8 bits)
  Red (8 bits)
```

```cpp
button->color(0xff000000);  // RGB: red
button->color(0x00ff0000);  // RGB: green
button->color(0x0000ff00);  // RGB: blue
button->color(0xffffff00);  // RGB: white
```

**Note**

If TrueColor is not available, any RGB colors will be set to the nearest entry in the colormap.

### 1.8.3.4 Line Dashes and Thickness

FLTK supports drawing of lines with different styles and widths. Full functionality is not available under Windows 95, 98, and Me due to the reduced drawing functionality these operating systems provide.

```cpp
void fl_line_style(int style, int width, char∗ dashes)
```

Set how to draw lines (the “pen”). If you change this it is your responsibility to set it back to the default with `fl_line_style(0)`.

**Note:** Because of how line styles are implemented on MS Windows systems, you *must* set the line style *after* setting the drawing color. If you set the color after the line style you will lose the line style settings!

`style` is a bitmask which is a bitwise-OR of the following values. If you don’t specify a dash type you will get a solid line. If you don’t specify a cap or join type you will get a system-defined default of whatever value is fastest.
width is the number of FLTK units thick to draw the lines. Zero results in the system-defined default, which on both X and Windows is somewhat different and nicer than 1.

dashes is a pointer to an array of dash lengths, measured in FLTK units. The first location is how long to draw a solid portion, the next is how long to draw the gap, then the solid, etc. It is terminated with a zero-length entry. A NULL pointer or a zero-length array results in a solid line. Odd array sizes are not supported and result in undefined behavior.

Note: The dashes array does not work under Windows 95, 98, or Me, since those operating systems do not support complex line styles.

1.8.3.5 Drawing Fast Shapes

These functions are used to draw almost all the FLTK widgets. They draw on exact pixel boundaries and are as fast as possible. Their behavior is duplicated exactly on all platforms FLTK is ported. It is undefined whether these are affected by the transformation matrix, so you should only call these while the matrix is set to the identity matrix (the default).

void fl_point(int x, int y)

Draw a single pixel at the given coordinates.

void fl_rectf(int x, int y, int w, int h)
void fl_rectf(int x, int y, int w, int h, Fl_Color c)

Color a rectangle that exactly fills the given bounding box.

void fl_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b)
Color a rectangle with "exactly" the passed \( r, g, b \) color. On screens with less than 24 bits of color this is done by drawing a solid-colored block using \texttt{fl\_draw\_image()} so that the correct color shade is produced.

```c
void fl_rect(int x, int y, int w, int h)
void fl_rect(int x, int y, int w, int h, Fl_Color c)
```

Draw a 1-pixel border inside this bounding box.

```c
void fl_rounded_rect(int x, int y, int w, int h, int radius)
void fl_rounded_rectf(int x, int y, int w, int h, int radius)
```

Draw an outlined or filled rectangle with rounded corners.

```c
void fl_line(int x, int y, int x1, int y1)
void fl_line(int x, int y, int x1, int y1, int x2, int y2)
```

Draw one or two lines between the given points.

```c
void fl_loop(int x, int y, int x1, int y1, int x2, int y2)
void fl_loop(int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)
```

Outline a 3 or 4-sided polygon with lines.

```c
void fl_polygon(int x, int y, int x1, int y1, int x2, int y2)
void fl_polygon(int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)
```

Fill a 3 or 4-sided polygon. The polygon must be convex.

```c
void fl_xyline(int x, int y, int x1)
void fl_xyline(int x, int y, int x1, int y2)
void fl_xyline(int x, int y, int x1, int y2, int x3)
```

Draw horizontal and vertical lines. A horizontal line is drawn first, then a vertical, then a horizontal.

```c
void fl_yxline(int x, int y, int y1)
void fl_yxline(int x, int y, int y1, int x2)
void fl_yxline(int x, int y, int y1, int x2, int y3)
```

Draw vertical and horizontal lines. A vertical line is drawn first, then a horizontal, then a vertical.

```c
void fl_arc(int x, int y, int w, int h, double a1, double a2)
void fl_pie(int x, int y, int w, int h, double a1, double a2)
```
Draw ellipse sections using integer coordinates. These functions match the rather limited circle drawing code provided by X and MS Windows. The advantage over using `fl_arc()` with floating point coordinates is that they are faster because they often use the hardware, and they draw much nicer small circles, since the small sizes are often hard-coded bitmaps.

If a complete circle is drawn it will fit inside the passed bounding box. The two angles are measured in degrees counter-clockwise from 3’oclock and are the starting and ending angle of the arc, \( a_2 \) must be greater or equal to \( a_1 \).

![Figure 1.26 fl_pie() and fl_arc()](image)

`fl_arc()` draws a series of lines to approximate the arc. Notice that the integer version of `fl_arc()` has a different number of arguments to the other `fl_arc()` function described later in this chapter.

`fl_pie()` draws a filled-in pie slice. This slice may extend outside the line drawn by `fl_arc()`; to avoid this use \( w-1 \) and \( h-1 \).

```c
void fl_scroll(int X, int Y, int W, int H, int dx, int dy, void (draw_area)(void, int, int, int, int), void* data) {
    // Scroll a rectangle and draw the newly exposed portions. The contents of the rectangular area is first shifted by dx and dy FLTK units. The callback is then called for every newly exposed rectangular area,
```

Generated by Doxygen
1.8 Drawing Things in FLTK

1.8.3.6 Drawing Complex Shapes

The complex drawing functions let you draw arbitrary shapes with 2-D linear transformations. The functionality matches that found in the Adobe® PostScript™ language. The exact pixels that are filled are less defined than for the fast drawing functions so that FLTK can take advantage of drawing hardware. On both X and MS Windows the transformed vertices are rounded to integers before drawing the line segments: this severely limits the accuracy of these functions for complex graphics, so use OpenGL when greater accuracy and/or performance is required.

```c
void fl_load_matrix(double a, double b, double c, double d, double x, double y);
void fl_load_identity();
```

Set the current transformation.

```c
void fl_push_matrix();
void fl_pop_matrix();
```

Save and restore the current transformation. The maximum depth of the stack is 32 entries.

```c
void fl_scale(double x, double y);
void fl_scale(double x);
void fl_translate(double x, double y);
void fl_rotate(double d);
void fl_mult_matrix(double a, double b, double c, double d, double x, double y);
```

Concatenate another transformation onto the current one. The rotation angle is in degrees (not radians) and is counter-clockwise.

```c
double fl_transform_x(double x, double y);
double fl_transform_y(double x, double y);
double fl_transform_dx(double x, double y);
double fl_transform_dy(double x, double y);
void fl_transformed_vertex(double xf, double yf);
```

Transform a coordinate or a distance using the current transformation matrix. After transforming a coordinate pair, it can be added to the vertex list without any further translations using `fl_transformed_vertex()`.

```c
void fl_begin_points();
void fl_end_points();
```

Start and end drawing a list of points. Points are added to the list with `fl_vertex()`.

```c
void fl_begin_line();
void fl_end_line();
```
Start and end drawing lines.

```c
void fl_begin_loop()
void fl_end_loop()
```

Start and end drawing a closed sequence of lines.

```c
void fl_begin_polygon()
void fl_end_polygon()
```

Start and end drawing a convex filled polygon.

```c
void fl_begin_complex_polygon()
void fl_gap()
void fl_end_complex_polygon()
```

Start and end drawing a complex filled polygon. This polygon may be concave, may have holes in it, or may be several disconnected pieces. Call `fl_gap()` to separate loops of the path. It is unnecessary but harmless to call `fl_gap()` before the first vertex, after the last one, or several times in a row.

`fl_gap()` should only be called between `fl_begin_complex_polygon()` and `fl_end_complex_polygon()`. To outline the polygon, use `fl_begin_loop()` and replace each `fl_gap()` with a `fl_end_loop(); fl_begin_loop()` pair.

**Note:** For portability, you should only draw polygons that appear the same whether "even/odd" or "non-zero" winding rules are used to fill them. Holes should be drawn in the opposite direction of the outside loop.

```c
void fl_vertex(double x, double y)
```

Add a single vertex to the current path.

```c
void fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)
```

Add a series of points on a Bézier curve to the path. The curve ends (and two of the points are) at $X_0, Y_0$ and $X_3, Y_3$.

```c
void fl_arc(double x, double y, double r, double start, double end)
```
Add a series of points to the current path on the arc of a circle; you can get elliptical paths by using scale and rotate before calling `fl_arc()`. The center of the circle is given by $x$ and $y$, and $r$ is its radius. `fl_arc()` takes `start` and `end` angles that are measured in degrees counter-clockwise from 3 o'clock. If `end` is less than `start` then it draws the arc in a clockwise direction.

`fl_arc()`, `fl_circle()` and `fl_draw()`: 65

![Figure 1.27 fl_arc(x,y,r,a1,a2)](image)

```c
void fl_circle(double x, double y, double r)
```

`fl_circle(x,y,r)` is equivalent to `fl_arc(x,y,r,0,360)` but may be faster. It must be the only thing in the path: if you want a circle as part of a complex polygon you must use `fl_arc()`.

**Note:** `fl_circle()` draws incorrectly if the transformation is both rotated and non-square scaled.

1.8.3.7 Drawing Text

All text is drawn in the current font. It is undefined whether this location or the characters are modified by the current transformation.

```c
void fl_draw(const char *, int x, int y)
void fl_draw(const char *, int n, int x, int y)
```

Draw a null-terminated string or an array of $n$ bytes starting at the given location. In both cases, the text must be UTF-8 encoded. Text is aligned to the left and to the baseline of the font. To align to the bottom, subtract `fl_descent()` from $y$. To align to the top, subtract `fl_descent()` and add `fl_height()`. This version of `fl_draw()` provides direct access to the text drawing function of the underlying OS. It does not apply any special handling to control characters.

```c
void fl_rtl_draw(const char *str, int n, int x, int y)
```
Draw a UTF-8 string of length n bytes right to left starting at the given x, y location.

```c
void fl_draw(const char* str, int x, int y, int w, int h, Fl_Align align, Fl_Image* img, int draw_symbols)
```

Fancy string drawing function which is used to draw all the labels. The string is formatted and aligned inside the passed box. Handles ‘\t’ and ‘\n’, expands all other control characters to ^X, and aligns inside or against the edges of the box described by x, y, w and h. See Fl_Widget::align() for values for align. The value FL_ALIGN_INSIDE is ignored, as this function always prints inside the box.

If img is provided and is not NULL, the image is drawn above or below the text as specified by the align value.

The draw_symbols argument specifies whether or not to look for symbol names starting with the "@" character.

```c
void fl_measure(const char *str, int& w, int& h, int draw_symbols)
```

Measure how wide and tall the string will be when printed by the fl_draw(...align) function. This includes leading/trailing white space in the string, kerning, etc.

If the incoming w is non-zero it will wrap to that width.

This will probably give unexpected values unless you have called fl_font() explicitly in your own code. Refer to the full documentation for fl_measure() for details on usage and how to avoid common pitfalls.

See also

- `fl_text_extents()` – measure the ‘inked’ area of a string
- `fl_width()` – measure the width of a string or single character
- `fl_height()` – measure the height of the current font
- `fl_descent()` – the height of the descender for the current font

```c
int fl_height()
```

Recommended minimum line spacing for the current font. You can also just use the value of size passed to fl_font().
1.8 Drawing Things in FLTK

See also

\texttt{fl\_text\_extents()}, \texttt{fl\_measure()}, \texttt{fl\_width()}, \texttt{fl\_descent()}

\textbf{int fl\_descent()}

Recommended distance above the bottom of a \texttt{fl\_height()} tall box to draw the text at so it looks centered vertically in that box.

\textbf{double fl\_width(const char* txt)}
\texttt{double fl\_width(const char* txt, int n)}
\texttt{double fl\_width(unsigned int unicode\_char)}

Return the width of a nul-terminated string, a sequence of \texttt{n} characters, or a single character in the current font.

See also

\texttt{fl\_measure()}, \texttt{fl\_text\_extents()}, \texttt{fl\_height()}, \texttt{fl\_descent()}

\textbf{void fl\_text\_extents(const char* txt, int& dx, int& dy, int& w, int& h)}

Determines the minimum dimensions of a nul-terminated string, i.e. the 'inked area'.

Given a string "\texttt{txt}" drawn using \texttt{fl\_draw(txt, x, y)} you would determine its extents in \texttt{FLTK units} on the display using \texttt{fl\_text\_extents(txt, dx, dy, wo, ho)} such that a bounding box that exactly fits around the inked area of the text could be drawn with \texttt{fl\_rect(x+dx, y+dy, wo, ho)}.

Refer to the full documentation for \texttt{fl\_text\_extents()} for details on usage.

See also

\texttt{fl\_measure()}, \texttt{fl\_width()}, \texttt{fl\_height()}, \texttt{fl\_descent()}

\textbf{const char* fl\_shortcut\_label(int shortcut)}

Unparse a shortcut value as used by \texttt{Fl\_Button} or \texttt{Fl\_Menu\_Item} into a human-readable string like "Alt+N". This only works if the shortcut is a character key or a numbered function key. If the shortcut is zero an empty string is returned. The return value points at a static buffer that is overwritten with each call.
1.8.3.8 Fonts

FLTK supports a set of standard fonts based on the Times, Helvetica/Arial, Courier, and Symbol typefaces, as well as custom fonts that your application may load. Each font is accessed by an index into a font table.

Initially only the first 16 faces are filled in. There are symbolic names for them: FL_HELVETICA, FL_TIMES, FL_COURIER, and modifier values FL_BOLD and FL_ITALIC which can be added to these, and FL_SYMBOL and FL_ZAPF_DINGBATS. Faces greater than 255 cannot be used in Fl_Widget labels, since Fl_Widget stores the index as a byte.

One important thing to note about 'current font' is that there are so many paths through the GUI event handling code as widgets are partially or completely hidden, exposed and then re-drawn and therefore you can not guarantee that 'current font' contains the same value that you set on the other side of the event loop. Your value may have been superseded when a widget was redrawn. You are strongly advised to set the font explicitly before you draw any text or query the width and height of text strings, etc.

```c
void fl_font(int face, int size)
```

Set the current font, which is then used by the routines described above. You may call this outside a draw context if necessary to call fl_width(), but on X this will open the display.

The font is identified by a `face` and a `size`. The size of the font is measured in FLTK units and not "points". Lines should be spaced `size` FLTK units apart or more.

```c
int fl_font()
int fl_size()
```

Returns the face and size set by the most recent call to `fl_font(a,b)`. This can be used to save/restore the font.

1.8.3.9 Character Encoding

FLTK 1.3 and later versions expect all text in Unicode UTF-8 encoding. UTF-8 is ASCII compatible for the first 128 characters. International characters are encoded in multibyte sequences.

FLTK expects individual characters, characters that are not part of a string, in UCS-4 encoding, which is also ASCII compatible, but requires 4 bytes to store a Unicode character.

FLTK can draw accurately any Unicode-supported script for which the system contains relevant fonts. Under X11 platforms, this requires to build the library with the OPTION_USE_PANGO CMake option turned On (or with configure --enable-pango).

Plain text drawing starting at a user-given coordinate is well supported by FLTK, including for right-to-left scripts. Further text-related operations (i.e., selection, formatting, input, and editing) are functional with left-to-right scripts only.

For more information about character encodings, see the chapter on Unicode and UTF-8 Support.
1.8.3.10 Drawing Overlays

These functions allow you to draw interactive selection rectangles without using the overlay hardware. FLTK will XOR a single rectangle outline over a window.

```c
void fl_overlay_rect(int x, int y, int w, int h)
void fl.overlay_clear()
```

`fl_overlay_rect()` draws a selection rectangle, erasing any previous rectangle by XOR-ing it first. `fl.overlay_clear()` will erase the rectangle without drawing a new one.

Using these functions is tricky. You should make a widget with both a `handle()` and `draw()` method. `draw()` should call `fl.overlay_clear()` before doing anything else. Your `handle()` method should call `window()->make_current()` and then `fl_overlay_rect()` after FL_DRAG events, and should call `fl.overlay_clear()` after a FL_RELEASE event.

1.8.4 Drawing Images

To draw images, you can either do it directly from data in your memory, or you can create a `Fl_Image` object. The advantage of drawing directly is that it is more intuitive, and it is faster if the image data changes more often than it is redrawn. The advantage of using the object is that FLTK will cache translated forms of the image (on X it uses a server pixmap) and thus redrawing is much faster.

1.8.4.1 Direct Image Drawing

The behavior when drawing images when the current transformation matrix is not the identity is not defined, so you should only draw images when the matrix is set to the identity.

```c
void fl_draw_image(const uchar *buf, int X, int Y, int W, int H, int D, int L)
void fl_draw_image_mono(const uchar *buf, int X, int Y, int W, int H, int D, int L)
```

Draw an 8-bit per color RGB or luminance image. The pointer points at the "r" data of the top-left pixel. Color data must be in r, g, b order. The top left corner is given by X and Y and the size of the image is given by W and H. D is the delta to add to the pointer between pixels, it may be any value greater or equal to 3, or it can be negative to flip the image horizontally. L is the delta to add to the pointer between lines (if 0 is passed it uses W * D), and may be larger than W * D to crop data, or negative to flip the image vertically.

It is highly recommended that you put the following code before the first `show()` of any window in your program to get rid of the dithering if possible:

```c
Fl::visual(FL_RGB);
```

Gray scale (1-channel) images may be drawn. This is done if `abs(D)` is less than 3, or by calling `fl_draw_image_mono()`. Only one 8-bit sample is used for each pixel, and on screens with different numbers of bits for red, green, and blue only gray colors are used. Setting D greater than 1 will let you display one channel of a color image.
Note: The X version does not support all possible visuals. If FLTK cannot draw the image in the current visual it will abort. FLTK supports any visual of 8 bits or less, and all common TrueColor visuals up to 32 bits.

typedef void (∗Fl_Draw_Image_Cb)(void ∗data, int x, int y, int w, uchar ∗buf)
void fl_draw_image(Fl_Draw_Image_Cb cb, void ∗data, int X, int Y, int W, int H, int D)
void fl_draw_image_mono(Fl_Draw_Image_Cb cb, void ∗data, int X, int Y, int W, int H, int D)

Call the passed function to provide each scan line of the image. This lets you generate the image as it is being drawn, or do arbitrary decompression of stored data, provided it can be decompressed to individual scan lines easily.

The callback is called with the void* user data pointer which can be used to point at a structure of information about the image, and the x, y, and w of the scan line desired from the image. 0,0 is the upper-left corner of the image, not X, Y. A pointer to a buffer to put the data into is passed. You must copy w pixels from scanline y, starting at pixel x, to this buffer.

Due to cropping, less than the whole image may be requested. So x may be greater than zero, the first y may be greater than zero, and w may be less than W. The buffer is long enough to store the entire W*D pixels, this is for convenience with some decompression schemes where you must decompress the entire line at once: decompress it into the buffer, and then if x is not zero, copy the data over so the x'th pixel is at the start of the buffer.

You can assume the y's will be consecutive, except the first one may be greater than zero.

If D is 4 or more, you must fill in the unused bytes with zero.

int fl_draw_pixmap(char const* data, int x, int y, Fl_Color bg)
int fl_draw_pixmap(const char const* cdata, int x, int y, Fl_Color bg)

Draws XPM image data, with the top-left corner at the given position. The image is dithered on 8-bit displays so you won't lose color space for programs displaying both images and pixmaps. This function returns zero if there was any error decoding the XPM data.

To use an XPM, do:

#include "foo.xpm"
...
fl_draw_pixmap(foo, X, Y);

Transparent colors are replaced by the optional Fl_Color argument. To draw with true transparency you must use the Fl_Pixmap class.

int fl_measure_pixmap(char const* data, int &, int &)
int fl_measure_pixmap(const char const* cdata, int &, int &)

An XPM image contains the dimensions in its data. This function finds and returns the width and height. The return value is non-zero if the dimensions were parsed ok and zero if there was any problem.
1.8.4.2 Direct Image Reading

FLTK provides a single function for reading from the current window or off-screen buffer into a RGB(A) image buffer.

uchar* fl_read_image(uchar *p, int X, int Y, int W, int H, int alpha)

Read a RGB(A) image from the current window or off-screen buffer. The \( p \) argument points to a buffer that can hold the image and must be at least \( W \cdot H \cdot 3 \) bytes when reading RGB images and \( W \cdot H \cdot 4 \) bytes when reading RGBA images. If NULL, \( \text{fl_read_image}() \) will create an array of the proper size which can be freed using delete[].

The \( \text{alpha} \) parameter controls whether an alpha channel is created and the value that is placed in the alpha channel. If 0, no alpha channel is generated.

1.8.4.3 Image Classes

FLTK provides a base image class called Fl_Image which supports creating, copying, and drawing images of various kinds, along with some basic color operations. Images can be used as labels for widgets using the image() and deimage() methods or drawn directly. Images can be drawn scaled to any size, independently from the size of the image’s data (see Fl_Image::scale()).

The Fl_Image class does almost nothing by itself, but is instead supported by three basic image types:

- Fl_Bitmap
- Fl_Pixmap
- Fl_RGB_Image

The Fl_Bitmap class encapsulates a mono-color bitmap image. The draw() method draws the image using the current drawing color.

The Fl_Pixmap class encapsulates a colormapped image. The draw() method draws the image using the colors in the file, and masks off any transparent colors automatically.

The Fl_RGB_Image class encapsulates a full-color (or grayscale) image with 1 to 4 color components. Images with an even number of components are assumed to contain an alpha channel that is used for transparency. The transparency provided by the draw() method is either a 24-bit blend against the existing window contents or a “screen door” transparency mask, depending on the platform and screen color depth.

char fl_can_do_alpha_blending()
fl_can_do_alpha_blending() will return 1, if your platform supports true alpha blending for RGBA images, or 0, if FLTK will use screen door transparency.

FLTK also provides several image classes based on the three standard image types for common file formats:

- Fl_GIF_Image
- Fl_Anim_GIF_Image
- Fl_JPEG_Image
- Fl_PNG_Image
- Fl_PNM_Image
- Fl_XBM_Image
- Fl_XPM_Image
- Fl_SVG_Image
- Fl_BMP_Image
- Fl_ICO_Image

Each of these image classes loads a named file of the corresponding format. The Fl_Shared_Image class can be used to load any type of image file - the class examines the file and constructs an image of the appropriate type.

Finally, FLTK provides a special image class called Fl_Tiled_Image to tile another image object in the specified area. This class can be used to tile a background image in a Fl_Group widget, for example.

virtual Fl_Image* Fl_Image::copy()
virtual Fl_Image* Fl_Image::copy(int W, int H) const

The copy() method creates a copy of the image. The second form specifies the new size of the image - the image is resized using the nearest-neighbor algorithm (this is the default).

Note
As of FLTK 1.3.3 the image resizing algorithm can be changed. See Fl_Image::RGB_scaling(Fl_RGB_Scaling method)

virtual void Fl_Image::draw(int x, int y, int w, int h, int ox, int oy)

The draw() method draws the image object. x, y, w, h indicates the destination rectangle. ox, oy, w, h is the source rectangle. This source rectangle is copied to the destination. The source rectangle may extend outside the image, i.e. ox and oy may be negative and w and h may be bigger than the image, and this area is left unchanged.

Note
See exceptions for Fl_Tiled_Image::draw() regarding arguments ox, oy, w, and h.

virtual void Fl_Image::draw(int x, int y)

Draws the image with the upper-left corner at x, y. This is the same as doing img->draw(x, y, img->w(), img->h(), 0, 0) where img is a pointer to any Fl_Image type.
1.8.5 Offscreen Drawing

Sometimes it can be very useful to generate a complex drawing in memory first and copy it to the screen at a later point in time. This technique can significantly reduce the amount of repeated drawing. Offscreen drawing functions are declared in `<FL/fl_draw.H>`.

`Fl_Double_Window` uses offscreen rendering to avoid flickering on systems that don't support double-buffering natively.

FLTK can draw into an offscreen buffer at any time. There is no need to wait for an `Fl_Widget::draw()` to occur.

Note

In FLTK 1.3.x and earlier versions all offscreen drawing functions described below were implemented as macros and created certain temporary variables to save context information. You needed to create local scope blocks with curly braces `{ ... }` if you used offscreen functions more than once in a function or method. This is no longer necessary since offscreen drawing is now implemented in real functions (no macros).

Example:

```c
Fl_Offscreen oscr = fl_create_offscreen(120, 120);
fl_begin_offscreen(oscr);
fl_color(FL_WHITE);
fl_rectf(0, 0, 120, 120);
fl_end_offscreen();
// other code here
fl_begin_offscreen(oscr);
fl_color(FL_BLACK);
fl_rectf(10, 10, 100, 100);
fl_end_offscreen();
// other code here
fl_delete_offscreen(oscr);
```

**Fl_Offscreen fl_create_offscreen(int w, int h)**

Create an RGB offscreen buffer containing as many pixels as in a screen area of size \( w, h \) FLTK units.

**void fl_delete_offscreen(Fl_Offscreen)**

Delete a previously created offscreen buffer. All drawings are lost.

**void fl_begin_offscreen(Fl_Offscreen)**

Send all subsequent drawing commands to this offscreen buffer.

**void fl_end_offscreen()**

Quit sending drawing commands to this offscreen buffer.

**void fl_copy_offscreen(int x, int y, int w, int h, Fl_Offscreen osrc, int srcx, int srcy)**

Copy a rectangular area of the size \( w \times h \) from \( srcx, srcy \) in the offscreen buffer into the current drawing surface at \( x, y \).

**void fl_rescale_offscreen(Fl_Offscreen &osrc)**

Adapts the offscreen's size in pixels to a changed value of the scale factor while keeping the offscreen's graphical content.
1.9 Handling Events

This chapter discusses the FLTK event model and how to handle events in your program or widget.

1.9.1 The FLTK Event Model

Every time a user moves the mouse pointer, clicks a button, or presses a key, an event is generated and sent to your application. Events can also come from other programs like the window manager.

Events are identified by the integer argument passed to a `handle()` method that overrides the `Fl_Widget::handle()` virtual method. Other information about the most recent event is stored in static locations and acquired by calling the `Fl::event_∗()` methods. This static information remains valid until the next event is read from the window system, so it is ok to look at it outside of the `handle()` method.

Event numbers can be converted to their actual names using the `fl_eventnames[]` array defined in `<FL/names.h>`; see next chapter for details.

In the next chapter, the `MyClass::handle()` example shows how to override the `Fl_Widget::handle()` method to accept and process specific events.

1.9.2 Mouse Events

1.9.2.1 FL_PUSH

A mouse button has gone down with the mouse pointing at this widget. You can find out what button by calling `Fl::event_button()`. You find out the mouse position by calling `Fl::event_x()` and `Fl::event_y()`.

A widget indicates that it "wants" the mouse click by returning non-zero from its `handle()` method, as in the `MyClass::handle()` example. It will then become the `Fl::pushed()` widget and will get `FL_DRAG` and the matching `FL_RELEASE` events. If `handle()` returns zero then FLTK will try sending the `FL_PUSH` to another widget.

1.9.2.2 FL_DRAG

The mouse has moved with a button held down. The current button state is in `Fl::event_state()`. The mouse position is in `Fl::event_x()` and `Fl::event_y()`.

In order to receive `FL_DRAG` events, the widget must return non-zero when handling `FL_PUSH`.

1.9.2.3 FL_RELEASE

A mouse button has been released. You can find out what button by calling `Fl::event_button()`.

In order to receive the `FL_RELEASE` event, the widget must return non-zero when handling `FL_PUSH`.

1.9.2.4 FL_MOVE

The mouse has moved without any mouse buttons held down. This event is sent to the `Fl::belowmouse()` widget.

In order to receive `FL_MOVE` events, the widget must return non-zero when handling `FL_ENTER`. 
1.9 Handling Events

1.9.2.5 FL_MOUSEWHEEL

The user has moved the mouse wheel. The \texttt{Fl::event\_dx()} and \texttt{Fl::event\_dy()} methods can be used to find the amount to scroll horizontally and vertically.

1.9.3 Focus Events

1.9.3.1 FL_ENTER

The mouse has been moved to point at this widget. This can be used for highlighting feedback. If a widget wants to highlight or otherwise track the mouse, it indicates this by returning non-zero from its \texttt{handle()} method. It then becomes the \texttt{Fl::belowmouse()} widget and will receive \texttt{FL\_MOVE} and \texttt{FL\_LEAVE} events.

1.9.3.2 FL_LEAVE

The mouse has moved out of the widget.

In order to receive the \texttt{FL\_LEAVE} event, the widget must return non-zero when handling \texttt{FL\_ENTER}.

1.9.3.3 FL_FOCUS

This indicates an attempt to give a widget the keyboard focus.

If a widget wants the focus, it should change itself to display the fact that it has the focus, and return non-zero from its \texttt{handle()} method. It then becomes the \texttt{Fl::focus()} widget and gets \texttt{FL\_KEYDOWN}, \texttt{FL\_KEYUP}, and \texttt{FL\_UNFOCUS} events.

The focus will change either because the window manager changed which window gets the focus, or because the user tried to navigate using tab, arrows, or other keys. You can check \texttt{Fl::event\_key()} to figure out why it moved. For navigation it will be the key pressed and for interaction with the window manager it will be zero.

1.9.3.4 FL_UNFOCUS

This event is sent to the previous \texttt{Fl::focus()} widget when another widget gets the focus or the window loses focus.
1.9.4 Keyboard Events

1.9.4.1 FL_KEYBOARD, FL_KEYDOWN, FL_KEYUP

A key was pressed (FL_KEYDOWN) or released (FL_KEYUP). FL_KEYBOARD is a synonym for FL_KEYDOWN, and both names are used interchangeably in this documentation.

The key can be found in Fl::event_key(). The text that the key should insert can be found with Fl::event_text() and its length is in Fl::event_length().

If you use the key, then handle() should return 1. If you return zero then FLTK assumes you ignored the key and will then attempt to send it to a parent widget. If none of them want it, it will change the event into a FL_SHORTCUT event. FL_KEYBOARD events are also generated by the character palette/map.

To receive FL_KEYBOARD events you must also respond to the FL_FOCUS and FL_UNFOCUS events by returning 1. This way FLTK knows whether to bother sending your widget keyboard events. (Some widgets don’t need them, e.g. Fl_Box.)

If you are writing a text-editing widget you may also want to call the Fl::compose() function to translate individual keystrokes into characters.

FL_KEYUP events are sent to the widget that currently has focus. This is not necessarily the same widget that received the corresponding FL_KEYDOWN event because focus may have changed between events.

Todo Add details on how to detect repeating keys, since on some X servers a repeating key will generate both FL_KEYUP and FL_KEYDOWN, such that to tell if a key is held down during FL_KEYUP or not.

1.9.4.2 FL_SHORTCUT

If the Fl::focus() widget is zero or ignores an FL_KEYBOARD event then FLTK tries sending this event to every widget it can, until one of them returns non-zero. FL_SHORTCUT is first sent to the Fl::belowmouse() widget, then its parents and siblings, and eventually to every widget in the window, trying to find an object that returns non-zero. FLTK tries really hard to not to ignore any keystrokes!

You can also make “global” shortcuts by using Fl::add_handler(). A global shortcut will work no matter what windows are displayed or which one has the focus.

1.9.5 Widget Events

1.9.5.1 FL_DEACTIVATE

This widget is no longer active, due to deactivate() being called on it or one of its parents. Please note that although active() may still return true for this widget after receiving this event, it is only truly active if active() is true for both it and all of its parents. (You can use active_r() to check this).

1.9.5.2 FL_ACTIVATE

This widget is now active, due to activate() being called on it or one of its parents.
1.9 Handling Events

1.9.5.3 FL_HIDE

This widget is no longer visible, due to hide() being called on it or one of its parents, or due to a parent window being minimized. Please note that although visible() may still return true for this widget after receiving this event, it is only truly visible if visible() is true for both it and all of its parents. (You can use visible_r() to check this).

1.9.5.4 FL_SHOW

This widget is visible again, due to show() being called on it or one of its parents, or due to a parent window being restored. A child Fl_Window will respond to this by actually creating the window if not done already, so if you subclass a window, be sure to pass FL_SHOW to the base class handle() method!

Note

The events in this chapter ("Widget Events"), i.e. FL_ACTIVATE, FL_DEACTIVATE, FL_SHOW, and FL_HIDE, are the only events deactivated and invisible widgets can usually get, depending on their states. Under certain circumstances, there may also be FL_LEAVE or FL_UNFOCUS events delivered to deactivated or hidden widgets.

1.9.6 Clipboard Events

1.9.6.1 FL_PASTE

You should get this event some time after you call Fl::paste(). The contents of Fl::event_text() is the text to insert and the number of characters is in Fl::event_length().

1.9.6.2 FL_SELECTIONCLEAR

The Fl::selection_owner() will get this event before the selection is moved to another widget. This indicates that some other widget or program has claimed the selection. Motif programs used this to clear the selection indication. Most modern programs ignore this.

1.9.7 Drag and Drop Events

FLTK supports drag and drop of text and files from any application on the desktop to an FLTK widget. Text is transferred using UTF-8 encoding.

See Fl::dnd() for drag and drop from an FLTK widget.

The drag and drop data is available in Fl::event_text() at the concluding FL_PASTE. On some platforms, the event text is also available for the FL_DND_* events, however application must not depend on that behavior because it depends on the protocol used on each platform.

FL_DND_* events cannot be used in widgets derived from Fl_Group or Fl_Window.
1.9.7.1 Dropped filenames

Files are received as a list of full path and file names.

- On some X11 platforms, files are received as a URL-encoded UTF-8 string, that is, non-ASCII bytes (and a few others such as space and %) are replaced by the 3 bytes "%XY" where XY are the byte's hexadecimal value. The `fl_decode_uri()` function can be used to transform in-place the received string into a proper UTF-8 string. On these platforms, strings corresponding to dropped files are further prepended by `file://` (or other prefixes such as `computer://`).

- Other X11 situations put all dropped filenames in a single line, separated by spaces.

- On non-X11 platforms, including Wayland, files dropped are received one pathname per line, with no `\n` after the last pathname.

1.9.7.2 FL_DND_ENTER

The mouse has been moved to point at this widget. A widget that is interested in receiving drag’n’drop data must return 1 to receive `FL_DND_DRAG`, `FL_DND_LEAVE` and `FL_DND_RELEASE` events.

1.9.7.3 FL_DND_DRAG

The mouse has been moved inside a widget while dragging data. A widget that is interested in receiving drag’n’drop data should indicate the possible drop position.

1.9.7.4 FL_DND_LEAVE

The mouse has moved out of the widget.

1.9.7.5 FL_DND_RELEASE

The user has released the mouse button dropping data into the widget. If the widget returns 1, it will receive the data in the immediately following `FL_PASTE` event.

1.9.8 Other events

1.9.8.1 FL_SCREEN_CONFIGURATION_CHANGED

Sent whenever the screen configuration changes (a screen is added/removed, a screen resolution is changed, screens are moved). Use `Fl::add_handler()` to be notified of this event.

1.9.8.2 FL_FULLSCREEN

The application window has been changed from normal to fullscreen, or from fullscreen to normal. If you are using a X window manager which supports Extended Window Manager Hints, this event will not be delivered until the change has actually happened.
1.9.9 Fl::event_*( ) methods

FLTK keeps the information about the most recent event in static storage. This information is good until the next event is processed. Thus it is valid inside handle( ) and callback( ) methods.

These are all trivial inline functions and thus very fast and small:

- Fl::event_button()
- Fl::event_clicks()
- Fl::event_dx()
- Fl::event_dy()
- Fl::event_inside()
- Fl::event_is_click()
- Fl::event_key()
- Fl::event_length()
- Fl::event_state()
- Fl::event_text()
- Fl::event_x()
- Fl::event_x_root()
- Fl::event_y()
- Fl::event_y_root()
- Fl::get_key()
- Fl::get_mouse()
- Fl::test_shortcut()

1.9.10 Event Propagation

Widgets receive events via the virtual handle( ) function. The argument indicates the type of event that can be handled. The widget must indicate if it handled the event by returning 1. FLTK will then remove the event and wait for further events from the host. If the widget's handle function returns 0, FLTK may redistribute the event based on a few rules.

Most events are sent directly to the handle( ) method of the Fl_Window that the window system says they belong to. The window (actually the Fl_Group that Fl_Window is a subclass of) is responsible for sending the events on to any child widgets. To make the Fl_Group code somewhat easier, FLTK sends some events (FL_DRAG, FL←_RELEASE, FL_KEYBOARD, FL_SHORTCUT, FL_UNFOCUS, and FL_LEAVE) directly to leaf widgets. These procedures control those leaf widgets:

- Fl::add_handler()
- Fl::belowmouse()
- Fl::focus()
- Fl::grab()
• Fl::modal()
• Fl::pushed()
• Fl::release() (deprecated, see Fl::grab(0))
• Fl_Widget::take_focus()

FLTK propagates events along the widget hierarchy depending on the kind of event and the status of the UI. Some events are injected directly into the widgets, others may be resent as new events to a different group of receivers.

Mouse click events are first sent to the window that caused them. The window then forwards the event down the hierarchy until it reaches the widget that is below the click position. If that widget uses the given event, the widget is marked “pushed” and will receive all following mouse motion (FL_DRAG) events until the mouse button is released.

Mouse motion (FL_MOVE) events are sent to the Fl::belowmouse() widget, i.e. the widget that returned 1 on the last FL_ENTER event.

Mouse wheel events are sent to the window that caused the event. The window propagates the event down the tree, first to the widget that is below the mouse pointer, and if that does not succeed, to all other widgets in the group. This ensures that scroll widgets work as expected with the widget furthest down in the hierarchy getting the first opportunity to use the wheel event, but also giving scroll bars, that are not directly below the mouse a chance.

Keyboard events are sent directly to the widget that has keyboard focus. If the focused widget rejects the event, it is resent as a shortcut event, first to the top-most window, then to the widget below the mouse pointer, propagating up the hierarchy to all its parents. Those send the event also to all widgets that are not below the mouse pointer. Now if that did not work out, the shortcut is sent to all registered shortcut handlers.

If we are still unsuccessful, the event handler flips the case of the shortcut letter and starts over. Finally, if the key is “escape”, FLTK sends a close event to the top-most window.

All other events are pretty much sent right away to the window that created the event.

Widgets can "grab" events. The grabbing window gets all events exclusively, but usually by the same rules as described above.

Windows can also request exclusivity in event handling by making the window modal.

### 1.9.11 FLTK Compose-Character Sequences

The character composition done by Fl_Input widget requires that you call the Fl::compose() function if you are writing your own text editor widget.

Currently, all characters made by single key strokes with or without modifier keys, or by system-defined character compose sequences (that can involve dead keys or a compose key) can be input. You should call Fl::compose() in case any enhancements to this processing are done in the future. The interface has been designed to handle arbitrary UTF-8 encoded text.

The following methods are provided for character composition:

• Fl::compose()
• Fl::compose_reset()

Under Mac OS X, FLTK "previews" partially composed sequences.
1.10 Adding and Extending Widgets

This chapter describes how to add your own widgets or extend existing widgets in FLTK.

1.10.1 Subclassing

New widgets are created by subclassing an existing FLTK widget, typically Fl_Widget for controls and Fl_Group for composite widgets.

A control widget typically interacts with the user to receive and/or display a value of some sort.

A composite widget holds a list of child widgets and handles moving, sizing, showing, or hiding them as needed. Fl_Group is the main composite widget class in FLTK, and all of the other composite widgets (Fl_Pack, Fl_Scroll, Fl_Tabs, Fl_Tile, Fl_Window, Fl_Flex, Fl_Grid, etc.) are subclasses of it.

You can also subclass other existing widgets to provide a different look or user-interface. For example, the button widgets are all subclasses of Fl_Button since they all interact with the user via a mouse button click. The only difference is the code that draws the face of the button.

1.10.2 Making a Subclass of Fl_Widget

Your subclasses can directly descend from Fl_Widget or any subclass of Fl_Widget. Fl_Widget has only four virtual methods, and overriding some or all of these may be necessary.

1.10.3 The Constructor

The constructor should have the following arguments:
MyClass(int x, int y, int w, int h, const char *label = 0);

This will allow the class to be used in FLUID without problems.

The constructor must call the constructor for the base class and pass the same arguments:
MyClass::MyClass(int x, int y, int w, int h, const char *label)
    : Fl_Widget(x, y, w, h, label) {
        // do initialization stuff...
    }

Fl_Widget's protected constructor sets x(), y(), w(), h(), and label() to the passed values and initializes the other instance variables to:
type(0);
box(FL_NO_BOX);
color(FL_BACKGROUND_COLOR);
selection_color(FL_BACKGROUND_COLOR);
labeltype(FL_NORMAL_LABEL);
labelstyle(FL_NORMAL_STYLE);
labelsize(FL_NORMAL_SIZE);
labelcolor(FL_FOREGROUND_COLOR);
align(FL_ALIGN_CENTER);
callback(default_callback,0);
flags(ACTIVE|VISIBLE);
image(0);
deimage(0);
1.10.4 Protected Methods of Fl_Widget

The following methods are provided for subclasses to use:

- clear_visible()
- damage()
- draw_box()
- draw_focus()
- draw_label()
- set_flag()
- set_visible()
- test_shortcut()
- type()

```cpp
void Fl_Widget::damage(uchar mask)
void Fl_Widget::damage(uchar mask, int x, int y, int w, int h)
uchar Fl_Widget::damage()
```

The first form indicates that a partial update of the object is needed. The bits in mask are OR’d into `damage()`. Your `draw()` routine can examine these bits to limit what it is drawing. The public method `Fl_Widget::redraw()` simply does `Fl_Widget::damage(FL_DAMAGE_ALL)`, but the implementation of your widget can call the public `damage(n)`.

The second form indicates that a region is damaged. If only these calls are done in a window (no calls to `damage(n)` then FLTK will clip to the union of all these calls before drawing anything. This can greatly speed up incremental displays. The mask bits are OR’d into `damage()` unless this is a `Fl_Window` widget.

The third form returns the bitwise-OR of all `damage(n)` calls done since the last `draw()`.

When redrawing your widgets you should look at the damage bits to see what parts of your widget need redrawing. The `handle()` method can then set individual damage bits to limit the amount of drawing that needs to be done:

```cpp
MyClass::handle(int event) {
    ...
    if (change_to_part1) damage(1);
    if (change_to_part2) damage(2);
    if (change_to_part3) damage(4);
}
```

```cpp
MyClass::draw() {
    if (damage() & FL_DAMAGE_ALL) {
        ... draw frame/box and other static stuff ...
    }
    if (damage() & (FL_DAMAGE_ALL | 1)) draw_part1();
    if (damage() & (FL_DAMAGE_ALL | 2)) draw_part2();
    if (damage() & (FL_DAMAGE_ALL | 4)) draw_part3();
}
```

**Todo** Clarify `Fl_Window::damage(uchar)` handling - seems confused/wrong? ORing value doesn’t match setting behavior in `Fl_Widget.h`!

```cpp
void Fl_Widget::draw_box() const
void Fl_Widget::draw_box(Fl_Boxtype t, Fl_Color c) const
```
The first form draws this widget's `box()`, using the dimensions of the widget. The second form uses `t` as the box type and `c` as the color for the box.

```cpp
void Fl_Widget::draw_focus()
void Fl_Widget::draw_focus(Fl_Boxtype t, int x, int y, int w, int h) const
```

Draws a focus box inside the widget's bounding box. The second form allows you to specify a different bounding box.

```cpp
void Fl_Widget::draw_label() const
void Fl_Widget::draw_label(int x, int y, int w, int h) const
void Fl_Widget::draw_label(int x, int y, int w, int h, Fl_Align align) const
```

The first form is the usual function for a `draw()` method to call to draw the widget's label. It does not draw the label if it is supposed to be outside the box (on the assumption that the enclosing group will draw those labels).

The second form uses the passed bounding box instead of the widget's bounding box. This is useful so "centered" labels are aligned with some feature, like a moving slider.

The third form draws the label anywhere. It acts as though `FL_ALIGN_INSIDE` has been forced on so the label will appear inside the passed bounding box. This is designed for parent groups to draw labels with.

```cpp
void Fl_Widget::set_flag(int c)
```

Calling `set_flag(SHORTCUT_LABEL)` modifies the behavior of `draw_label()` so that `&` characters cause an underscore to be printed under the next letter.

```cpp
void Fl_Widget::set_visible()
void Fl_Widget::clear_visible()
```

Fast inline versions of `Fl_Widget::hide()` and `Fl_Widget::show()`. These do not send the `FL_HIDE` and `FL_SHOW` events to the widget.

```cpp
int Fl_Widget::test_shortcut()
static int Fl_Widget::test_shortcut(const char *s)
```

The first version tests `Fl_Widget::label()` against the current event (which should be a `FL_SHORTCUT` event). If the label contains a `&` character and the character after it matches the keypress, this returns true. This returns false if the `SHORTCUT_LABEL` flag is off, if the label is `NULL`, or does not have a `&` character in it, or if the keypress does not match the character.
The second version lets you do this test against an arbitrary string.

uchar Fl_Widget::type() const
void Fl_Widget::type(uchar t)

The property Fl_Widget::type() can return an arbitrary 8-bit identifier, and can be set with the protected method type(uchar t). This value had to be provided for Forms compatibility, but you can use it for any purpose you want. Try to keep the value less than 100 to not interfere with reserved values.

FLTK does not use RTTI (Run Time Typing Information) to enhance portability. But this may change in the near future if RTTI becomes standard everywhere.

If you don’t have RTTI you can use the clumsy FLTK mechanism, by having type() use a unique value. These unique values must be greater than the symbol FL_RESERVED_TYPE (which is 100) and less than FL_WINDOW (unless you make a subclass of Fl_Window). Look through the header files for FL_RESERVED_TYPE to find an unused number. If you make a subclass of Fl_Window you must use FL_WINDOW + n (where n must be in the range 1 to 7).

### 1.10.5 Handling Events

The virtual method Fl_Widget::handle(int event) is called to handle each event passed to the widget. It can:

- Change the state of the widget.
- Call Fl_Widget::redraw() if the widget needs to be redisplayed.
- Call Fl_Widget::damage(uchar c) if the widget needs a partial-update (assuming you provide support for this in your draw() method).
- Call Fl_Widget::do_callback() if a callback should be generated.
- Call Fl_Widget::handle() on child widgets.

Events are identified by the integer argument. Other information about the most recent event is stored in static locations and acquired by calling the Fl::event_∗() methods. This information remains valid until another event is handled.

Here is a sample handle() method for a widget that acts as a pushbutton and also accepts the keystroke ‘x’ to cause the callback:

```c
int MyClass::handle(int event) {
  switch(event) {
    case FL_PUSH:     
      highlight = 1;
      redraw();
      return 1;
    case FL_DRAG: {
      int t = Fl::event_inside(this);
      if (t != highlight) {
        highlight = t;
        redraw();
      }
    } return 1;
    case FL_RELEASE: 
      return 1;
  }
}
```
if (highlight) {
    highlight = 0;
    redraw();
    do_callback();
    // never do anything after a callback, as the callback
    // may delete the widget!
}
return 1;

case FL_SHORTCUT:
    if (Fl::event_key() == 'x') {
        do_callback();
        return 1;
    }
    return 0;
default:
    return Fl_Widget::handle(event);
}

You must return non-zero if your handle() method uses the event. If you return zero, the parent widget will try
sending the event to another widget.

For debugging purposes, event numbers can be printed as their actual event names using the fl_eventnames[] array,
e.g.:

```
#include <FL/names.h> // defines fl_eventnames[]
[..]
int MyClass::handle(int e) {
    printf("Event was %s (%d)\n", fl_eventnames[e], e); // e.g. "Event was FL_PUSH (1)"
   [..]
```

### 1.10.6 Drawing the Widget

The draw() virtual method is called when FLTK wants you to redraw your widget. It will be called if and only if
damage() is non-zero, and damage() will be cleared to zero after it returns. The draw() method should be
declared protected so that it can't be called from non-drawing code.

The damage() value contains the bitwise-OR of all the damage(n) calls to this widget since it was last drawn.
This can be used for minimal update, by only redrawing the parts whose bits are set. FLTK will turn on the FL_DAMAGE_ALL bit if it thinks the entire widget must be redrawn, e.g. for an expose event.

Expose events (and the damage(mask,x,y,w,h) function described above) will cause draw() to be called
with FLTK's clipping turned on. You can greatly speed up redrawing in some cases by testing fl_not_clipped(x,y,w,h) or fl_clip_box() and skipping invisible parts.

Besides the protected methods described above, FLTK provides a large number of basic drawing functions, which
are described in the chapter Drawing Things in FLTK.

### 1.10.7 Resizing the Widget

The resize(x,y,w,h) method is called when the widget is being resized or moved. The arguments are the
new position, width, and height. x(), y(), w(), and h() still remain the old size. You must call resize() on
your base class with the same arguments to get the widget size to actually change.

This should not call redraw(), at least if only the x() and y() change. This is because composite widgets like Fl_Scroll may have a more efficient way of drawing the new position.
1.10.8 Making a Composite Widget

A "composite" widget contains one or more "child" widgets. To make a composite widget you should subclass Fl_Group. It is possible to make a composite object that is not a subclass of Fl_Group, but you'll have to duplicate the code in Fl_Group anyways.

Instances of the child widgets may be included in the parent:

```cpp
class MyClass : public Fl_Group {
   Fl_Button the_button;
   Fl_Slider the_slider;
   ...
};
```

The constructor has to initialize these instances. They are automatically added to the group, since the Fl_Group constructor does Fl_Group::begin(). Don't forget to call Fl_Group::end() or use the Fl_End pseudo-class:

```cpp
MyClass::MyClass(int x, int y, int w, int h) :
   Fl_Group(x, y, w, h),
   the_button(x + 5, y + 5, 100, 20),
   the_slider(x, y + 50, w, 20)
{
   ...(you could add dynamically created child widgets here)...
   end(); // don't forget to do this!
}
```

The child widgets need callbacks. These will be called with a pointer to the children, but the widget itself may be found in the parent() pointer of the child. Usually these callbacks can be static private methods, with a matching private method:

```cpp
void MyClass::static_slider_cb(Fl_Widget* v, void *) { // static method
   ((MyClass*)(v->parent())->slider_cb();
}
void MyClass::slider_cb() { // normal method
   use(the_slider->value());
}
```

If you make the handle() method, you can quickly pass all the events to the children using the Fl_Group::handle() method. You don't need to override handle() if your composite widget does nothing other than pass events to the children:

```cpp
int MyClass::handle(int event) {
   if (Fl_Group::handle(event)) return 1;
   // handle events that children don't want ...
}
```

If you override draw() you need to draw all the children. If redraw() or damage() is called on a child, damage(FL_DAMAGE_CHILD) is done to the group, so this bit of damage() can be used to indicate that a child needs to be drawn. It is fastest if you avoid drawing anything else in this case:

```cpp
int MyClass::draw() {
   Fl_Widget *const* a = array();
   if (damage() == FL_DAMAGE_CHILD) { // only redraw some children
      for (int i = children(); i--; a++) update_child(**a);
   } else { // total redraw
      // draw background graphics ...
      // now draw all the children atop the background:
      for (int i = children_; i--; a++) { // draw children...
         draw_child(**a);
         draw_outside_label(**a); // you may not need to do this
      }
   }
}
```

Fl_Group provides some protected methods to make drawing easier:

- draw_child()
- draw_children()
- draw_outside_label()
- update_child()

```cpp
void Fl_Group::draw_child(Fl_Widget &widget) const
```
This will force the child’s `damage()` bits all to one and call `draw()` on it, then clear the `damage()`. You should call this on all children if a total redraw of your widget is requested, or if you draw something (like a background box) that damages the child. Nothing is done if the child is not `visible()` or if it is clipped.

```cpp
void Fl_Group::draw_children()
```

A convenience function that draws all children of the group. This is useful if you derived a widget from `Fl_Group` and want to draw a special border or background. You can call `draw_children()` from the derived `draw()` method after drawing the box, border, or background.

```cpp
void Fl_Group::draw_outside_label(const Fl_Widget &widget) const
```

Draw the labels that are not drawn by `draw_label()`. If you want more control over the label positions you might want to call `child->draw_label(x, y, w, h, a)`.

```cpp
void Fl_Group::update_child(Fl_Widget& widget) const
```

Draws the child only if its `damage()` is non-zero. You should call this on all the children if your own damage is equal to `FL_DAMAGE_CHILD`. Nothing is done if the child is not `visible()` or if it is clipped.

### 1.10.9 Cut and Paste Support

FLTK provides routines to cut and paste UTF-8 encoded text between applications:

- `Fl::copy()`
- `Fl::paste()`
- `Fl::selection()`
- `Fl::selection_owner()`

It is also possible to copy and paste image data between applications:

- `Fl_Copy_Surface`
- `Fl::clipboard_contains()`
- `Fl::paste()`

It may be possible to cut/paste other kinds of data by using `Fl::add_handler()`. Note that handling events beyond those provided by FLTK may be operating system specific. See Operating System Issues for more details.
1.10.10 Drag And Drop Support

FLTK provides routines to drag and drop UTF-8 encoded text between applications:

Drag’n’drop operations are initiated by copying data to the clipboard and calling the function Fl::dnd().

Drop attempts are handled via the following events, already described under Drag and Drop Events in a previous chapter:

- FL_DND_ENTER
- FL_DND_DRAG
- FL_DND_LEAVE
- FL_DND_RELEASE
- FL_PASTE

1.10.11 Making a subclass of Fl_Window

You may want your widget to be a subclass of Fl_Window, Fl_Double_Window, or Fl_Gl_Window. This can be useful if your widget wants to occupy an entire window, and can also be used to take advantage of system-provided clipping, or to work with a library that expects a system window ID to indicate where to draw.

Subclassing Fl_Window is almost exactly like subclassing Fl_Group, and in fact you can easily switch a subclass back and forth. Watch out for the following differences:

1. Fl_Window is a subclass of Fl_Group so make sure your constructor calls end() unless you actually want children added to your window.
2. When handling events and drawing, the upper-left corner is at 0,0, not x(),y() as in other Fl_Widget’s. For instance, to draw a box around the widget, call draw_box(0,0,w(),h()), rather than draw←
   box(x(),y(),w(),h()).

You may also want to subclass Fl_Window in order to get access to different visuals or to change other attributes of the windows. See the Operating System Issues chapter for more information.

1.11 Using OpenGL

This chapter discusses using FLTK for your OpenGL applications.

1.11.1 Using OpenGL in FLTK

The easiest way to make an OpenGL display is to subclass Fl_Gl_Window. Your subclass must implement a draw() method which uses OpenGL calls to draw the display. Your main program should call redraw() when the display needs to change, and (somewhat later) FLTK will call draw().

With a bit of care you can also use OpenGL to draw into normal FLTK windows (see Using OpenGL in Normal FLTK Windows below). This allows you to use Gouraud shading for drawing your widgets. To do this you use the gl_start() and gl_finish() functions around your OpenGL code.

You must include FLTK’s <FL/gl.h> header file. It will include the file <GL/gl.h> (on macOS: <OpenGL/gl.h>), define some extra drawing functions provided by FLTK, and include the <windows.h> header file needed by Windows applications.

Some simple coding rules (see OpenGL and support of HighDPI displays) allow to write cross-platform code that will support OpenGL run on HighDPI displays (including the 'retina' displays of Apple hardware).
1.11.2 Making a Subclass of Fl_Gl_Window

To make a subclass of Fl_Gl_Window, you must provide:

- A class definition.
- A `draw()` method.
- A `handle()` method if you need to receive input from the user.

If your subclass provides static controls in the window, they must be redrawn whenever the FL_DAMAGE_ALL bit is set in the value returned by `damage()`. For double-buffered windows you will need to surround the drawing code with the following code to make sure that both buffers are redrawn:

```c
#ifndef MESA
glDrawBuffer(GL_FRONT_AND_BACK);
#endif // !MESA
... draw stuff here ...
#ifndef MESA
glDrawBuffer(GL_BACK);
#endif // !MESA
```

**Note:**
If you are using the Mesa graphics library, the call to `glDrawBuffer()` is not required and will slow down drawing considerably. The preprocessor instructions shown above will optimize your code based upon the graphics library used.

1.11.2.1 Defining the Subclass

To define the subclass you just subclass the Fl_Gl_Window class:

```c
class MyWindow : public Fl_Gl_Window {
protected:
    void draw();
    int handle(int);
public:
    MyWindow(int X, int Y, int W, int H, const char *L)
        : Fl_Gl_Window(X, Y, W, H, L) {}
};
```

The `draw()` and `handle()` methods are described below. Like any widget, you can include additional private and public data in your class (such as scene graph information, etc.)

1.11.2.2 The `draw()` Method

The `draw()` method is where you actually do your OpenGL drawing:

```c
void MyWindow::draw() {
    if (!valid()) {
        ... set up projection, viewport, etc ...  
        ... window size is in w() and h()...
        ... valid() is turned on by FLTK after draw() returns
    }
    ... draw ...
}
```
1.11.2.3 The handle() Method

The `handle()` method handles mouse and keyboard events for the window:

```cpp
int MyWindow::handle(int event) {
    switch(event) {
    case FL_PUSH:
        // mouse down event ...
        return 1;
    case FL_DRAG:
        // mouse moved while down event ...
        return 1;
    case FL_RELEASE:
        // mouse up event ...
        return 1;
    case FL_FOCUS :
        case FL_UNFOCUS :
            // Return 1 if you want keyboard events, 0 otherwise
            return 1;
    case FL_KEYBOARD:
        // keypress, key is in Fl::event_key(), ascii in Fl::event_text()
        return 1 if you understand/use the keyboard event, 0 otherwise...
        return 1;
    case FL_SHORTCUT:
        // shortcut, key is in Fl::event_key(), ascii in Fl::event_text()
        return 1 if you understand/use the shortcut event, 0 otherwise...
        return 1;
    default:
        // pass other events to the base class...
        return Fl_Gl_Window::handle(event);
    }
}
```

When `handle()` is called, the OpenGL context is not set up! If your display changes, you should call `redraw()` and let `draw()` do the work. Don't call any OpenGL drawing functions from inside `handle()`!

You can call some OpenGL stuff like hit detection and texture loading functions by doing:

```cpp
case FL_PUSH:
    make_current(); // make OpenGL context current
    if (!valid()) {
        ... set up projection exactly the same as draw ...

        valid(1); // stop it from doing this next time
    }
    ... ok to call NON-DRAWING OpenGL code here, such as hit detection, loading textures, etc...
```

Your main program can now create one of your windows by doing `new MyWindow(...)`.

You can also use your new window class in FLUID by:

1. Putting your class definition in a `MyWindow.H` file.
2. Creating a `Fl_Box` widget in FLUID.
3. In the widget panel fill in the "class" field with `MyWindow`. This will make FLUID produce constructors for your new class.
4. In the "Extra Code" field put `#include "MyWindow.H"`, so that the FLUID output file will compile.

You must put `glwindow->show()` in your main code after calling `show()` on the window containing the OpenGL window.
1.11.3 OpenGL and support of HighDPI displays

HighDPI displays (including the so-called 'retina' displays of Apple hardware) are supported by FLTK in such a way that 1 unit of an FLTK quantity (say, the value given by `Fl_Gl_Window::w()` corresponds to more than 1 pixel on the display. Conversely, when a program specifies the width and height of the OpenGL viewport, it is necessary to use an API that returns quantities expressed in pixels. That can be done as follows:

```cpp
Fl_Gl_Window *glw = ...;
glViewport(0, 0, glw->pixel_w(), glw->pixel_h());
```

which makes use of the `Fl_Gl_Window::pixel_w()` and `Fl_Gl_Window::pixel_h()` methods giving the size in pixels of an `Fl_Gl_Window` that is potentially mapped to a HighDPI display. Method `Fl_Gl_Window::pixels_per_unit()` can also be useful in this context.

Note
A further coding rule is necessary to properly support retina displays and OpenGL under macOS (see OpenGL and 'retina' displays)

1.11.4 Using OpenGL in Normal FLTK Windows

Note
Drawing both with OpenGL and Quartz in a normal FLTK window is not possible with the macOS platform. This technique is therefore not useful under macOS because it permits nothing more than what is possible with class `Fl_Gl_Window`.

You can put OpenGL code into the `draw()` method, as described in Drawing the Widget in the previous chapter, or into the code for a boxtype or other places with some care.

Most importantly, before you show any windows, including those that don't have OpenGL drawing, you must initialize FLTK so that it knows it is going to use OpenGL. You may use any of the symbols described for `Fl_Gl_Window::mode()` to describe how you intend to use OpenGL:

```cpp
Fl::gl_visual(FL_RGB);
```

You can then put OpenGL drawing code anywhere you can draw normally by surrounding it with `gl_start()` and `gl_finish()` to set up, and later release, an OpenGL context with an orthographic projection so that 0,0 is the lower-left corner of the window and each pixel is one unit. The current clipping is reproduced with OpenGL `glScissor()` commands. These functions also synchronize the OpenGL graphics stream with the drawing done by other X, Windows, or FLTK functions.

```cpp
gl_start();
... put your OpenGL code here ...
gl_finish();
```

The same context is reused each time. If your code changes the projection transformation or anything else you should use `glPushMatrix()` and `glPopMatrix()` functions to put the state back before calling `gl_finish()`.

You may want to use `Fl_Window::current() -> h()` to get the drawable height so that you can flip the Y coordinates.

Unfortunately, there are a bunch of limitations you must adhere to for maximum portability:

- You must choose a default visual with `Fl::gl_visual()`.
- You cannot pass `FL_DOUBLE` to `Fl::gl_visual()`.
- You cannot use `Fl_Double_Window` or `Fl_Overlay_Window`.

Do not call `gl_start()` or `gl_finish()` when drawing into an `Fl_Gl_Window`!
1.11.5 Using FLTK widgets in OpenGL Windows

FLTK widgets can be added to Fl_Gl_Window just as they would be added to Fl_Window. They are rendered as an overlay over the user defined OpenGL graphics using 'fl_..' graphics calls that are implemented in GL.

Fl_Gl_Window does not add subsequent widgets as children by default as Fl_Window does. Call myGlWindow->begin() after creating the GL window to automatically add following widgets. Remember to call myGlWindow->end().

```cpp
class My_Gl_Window : public Fl_Gl_Window {
...
  void draw();
...};
...
myGlWindow = new My_Gl_Window(0, 0, 500, 500);
myGlWindow->begin();
myButton = new Fl_Button(10, 10, 120, 24, "Hello!");
myGlWindow->end();
...

void My_Gl_Window::draw() {
  // ... user GL drawing code
  Fl_Gl_Window::draw(); // Draw FLTK child widgets.
}
```

Users can draw into the overlay by using GL graphics calls as well as all fl_... graphics calls from the "Drawing Fast Shapes" section.

```cpp
void My_Gl_Window::draw() {
  // ... user GL drawing code
  Fl_Gl_Window::draw_begin(); // Set up 1:1 projection
  Fl_Window::draw(); // Draw FLTK children
  fl_color(FL_RED);
  fl_rect(10, 10, 100, 100);
  Fl_Gl_Window::draw_end(); // Restore GL state
}
```

Widgets can be drawn with transparencies by assigning an alpha value to a colormap entry and using that color in the widget.

```cpp
Fl::set_color(FL_FREE_COLOR, 255, 255, 0, 127); // 50% transparent yellow
myGlWindow = new My_Gl_Window(0, 0, 500, 500);
myGlWindow->begin();
myButton = new Fl_Button(10, 10, 120, 24, "Hello!");
myButton->box(FL_BORDER_BOX);
myButton->color(FL_FREE_COLOR);
myGlWindow->end();
```

Transparencies can also be set directly when drawing. This can be used to create custom box types and RGB overlay drawings with an alpha channel.

```cpp
fl_color(0, 255, 0, 127); // 50% transparent green
fl_rectf(10, 10, 100, 100);
fl_color(FL_RED); // back to opaque red
fl_rect(20, 20, 80, 80);
```

1.11.6 OpenGL Drawing Functions

FLTK provides some useful OpenGL drawing functions. They can be freely mixed with any OpenGL calls, and are defined by including `<FL/gl.h>` which you should include instead of the OpenGL header `<GL/gl.h>`.

```cpp
void gl_color(Fl_Color)

Sets the current OpenGL color to a FLTK color. For color-index modes it will use fl_xpixel(c), which is only right if this window uses the default colormap!

void gl_rect(int x, int y, int w, int h)
void gl_rectf(int x, int y, int w, int h)
```
Outlines or fills a rectangle with the current color. If Fl_Gl_Window::ortho() has been called, then the rectangle will exactly fill the pixel rectangle passed.

```c
void gl_font(Fl_Font fontid, int size)
```

Sets the current OpenGL font to the same font you get by calling fl_font().

```c
int gl_height()
int gl_descent()
float gl_width(const char *s)
float gl_width(const char *s, int n)
float gl_width(uchar c)
```

Returns information about the current OpenGL font.

```c
void gl_draw(const char *s)
void gl_draw(const char *s, int n)
```

Draws a nul-terminated string or an array of $n$ characters in the current OpenGL font at the current raster position.

```c
void gl_draw(const char *s, int x, int y)
void gl_draw(const char *s, int n, int x, int y)
void gl_draw(const char *s, float x, float y)
void gl_draw(const char *s, int n, float x, float y)
```

Draws a nul-terminated string or an array of $n$ characters in the current OpenGL font at the given position.

```c
void gl_draw(const char *s, int x, int y, int w, int h, Fl_Align)
```

Draws a string formatted into a box, with newlines and tabs expanded, other control characters changed to \^X, and aligned with the edges or center. Exactly the same output as fl_draw().

### 1.11.7 Speeding up OpenGL

Performance of Fl_Gl_Window may be improved on some types of OpenGL implementations, in particular MESA and other software emulators, by setting the GL_SWAP_TYPE environment variable. This variable declares what is in the backbuffer after you do a swapbuffers.

- `setenv GL_SWAP_TYPE COPY`

  This indicates that the back buffer is copied to the front buffer, and still contains its old data. This is true of many hardware implementations. Setting this will speed up emulation of overlays, and widgets that can do partial update can take advantage of this as `damage()` will not be cleared to -1.

- `setenv GL_SWAP_TYPE NODAMAGE`

  This indicates that nothing changes the back buffer except drawing into it. This is true of MESA and Win32 software emulation and perhaps some hardware emulation on systems with lots of memory.

- All other values for GL_SWAP_TYPE, and not setting the variable, cause FLTK to assume that the back buffer must be completely redrawn after a swap.

This is easily tested by running the gl_overlay demo program and seeing if the display is correct when you drag another window over it or if you drag the window off the screen and back on. You have to exit and run the program again for it to see any changes to the environment variable.
1.11.8 Using OpenGL Optimizer with FLTK

OpenGL Optimizer is a scene graph toolkit for OpenGL available from Silicon Graphics for IRIX and Microsoft Windows. It allows you to view large scenes without writing a lot of OpenGL code.

OptimizerWindow Class Definition

To use OpenGL Optimizer with FLTK you'll need to create a subclass of Fl_Gl_Widget that includes several state variables:

class OptimizerWindow : public Fl_Gl_Window {
    csContext *context_; // Initialized to 0 and set by draw()...
    csDrawAction *draw_action_; // Draw action...
    csGroup *scene_; // Scene to draw...
    csCamara *camera_; //Viewport for scene...

    void draw();

    public:
        OptimizerWindow(int X, int Y, int W, int H, const char *L)
            : Fl_Gl_Window(X, Y, W, H, L) {
            context_ = (csContext *)0;
            draw_action_ = (csDrawAction *)0;
            scene_ = (csGroup *)0;
            camera_ = (csCamera *)0;
        }

        void scene(csGroup *g) { scene_ = g; redraw(); }
        void camera(csCamera *c) {
            camera_ = c;
            if (context_) { draw_action_->setCamera(camera_);
                camera_->draw(draw_action_);
                redraw();
            }
        }
};

The camera() Method

The camera() method sets the camera (projection and viewpoint) to use when drawing the scene. The scene is redrawn after this call.

The draw() Method
The `draw()` method performs the needed initialization and does the actual drawing:

```cpp
void OptimizerWindow::draw() {
    if (!context_) {
        // This is the first time we've been asked to draw; create the
        // Optimizer context for the scene...
        #ifdef _WIN32
            context_ = new csContext((HDC)fl_getHDC());
            context_->ref();
            context_->makeCurrent((HDC)fl_getHDC());
        #else
            context_ = new csContext(fl_display, fl_visual);
            context_->ref();
            context_->makeCurrent(fl_display, fl_window);
        #endif // _WIN32
        ... perform other context setup as desired ...
        // Then create the draw action to handle drawing things...
        draw_action_ = new csDrawAction;
        if (camera_) {
            draw_action_->setCamera(camera_);
            camera_->draw(draw_action_);
        }
    } else {
        #ifdef _WIN32
            context_->makeCurrent((HDC)fl_getHDC());
        #else
            context_->makeCurrent(fl_display, fl_window);
        #endif // _WIN32
    }
    if (!valid()) {
        // Update the viewport for this context...
        context_->setViewport(0, 0, w(), h());
    }
    // Clear the window...
    context_->clear(csContext::COLOR_CLEAR | csContext::DEPTH_CLEAR,
                    0.0f, // Red
                    0.0f, // Green
                    0.0f, // Blue
                    1.0f); // Alpha
    // Then draw the scene (if any)...
    if (scene_)
        draw_action_->apply(scene_);
}
```

The `scene()` method sets the scene to be drawn. The scene is a collection of 3D objects in a `csGroup`. The scene is redrawn after this call.

### 1.11.9 Using OpenGL 3.0 (or higher versions)

The examples subdirectory contains OpenGL3test.cxx, a toy program showing how to use OpenGL 3.0 (or higher versions) with FLTK in a cross-platform fashion. It contains also OpenGL3-glut-test.cxx which shows how to use FLTK’s GLUT compatibility and OpenGL 3.

To access OpenGL 3.0 (or higher versions), use the `FL_OPENGL3` flag when calling `Fl_Gl_Window::mode(int a)` or `glutInitDisplayMode()`.

On the Windows and Linux platforms, FLTK creates contexts implementing the highest OpenGL version supported by the hardware. Such contexts may also be compatible with lower OpenGL versions. Access to functions from OpenGL versions above 1.1 requires to load function pointers at runtime on these platforms. FLTK recommends to use the GLEW library to perform this. It is therefore necessary to install the GLEW library (see below).

On the macOS platform, MacOS 10.7 or above is required; GLEW is possible but not necessary. FLTK creates contexts for OpenGL versions 1 and 2 without the `FL_OPENGL3` flag and for OpenGL versions 3.2 and above (but not below) with it.
GLEW installation (Linux and Windows platforms)

FLTK needs a header file, GL/glew.h, and a library, libGLEW.* or equivalent, to support OpenGL 3 and above.

These can be obtained for most Linux distributions by installing package libglew-dev.

For the Windows platform:

- the header and a Visual Studio static library (glew32.lib) can be downloaded from http://glew.sourceforge.net/;
- a MinGW-style static library (libglew32.a) can be built from source (same web site) with the make command. Alternatively, pre-built files are available for these architectures:
  - x86: download files glew.h and libglew32.a;
  - x86_64: install GLEW as an MSYS2 package with command: pacman -S mingw-w64-x86_64-glew

Source-level changes for OpenGL 3:

- Put this in all OpenGL-using source files (instead of, or before if needed, #include <FL/gl.h>, and before #include <FL/glut.h> if you use GLUT):
  #if defined(__APPLE__)
  # include <OpenGL/gl3.h> // defines OpenGL 3.0+ functions
  #else
  # if defined(_WIN32)
  # define GLEW_STATIC 1
  # endif
  # include <GL/glew.h>
  #endif

- Add the FL_OPENGL3 flag when calling Fl_Gl_Window::mode(int a) or glutInitDisplayMode().

- Put this in the handle(int event) member function of the first to be created among your Fl_Gl-Window-derived classes:
  #ifndef __APPLE__
  static int first = 1;
  if (first && event == FL_SHOW && shown()) {
    first = 0;
    make_current();
    glewInit(); // defines pters to functions of OpenGL V 1.2 and above
  }
  #endif

- Alternatively, if you use GLUT, put
  #ifndef __APPLE__
  glewInit(); // defines pters to functions of OpenGL V 1.2 and above
  #endif

after the first glutCreateWindow() call.

If GLEW is installed on the Mac OS development platform, it is possible to use the same code for all platforms, with one exception: put

#ifdef __APPLE__
glewExperimental = GL_TRUE;
#endif

before the glewinit() call.

Testing for success of the glewinit() call

Testing whether the glewinit() call is successful is to be done as follows:

#include <FL/Fl.H> // for Fl::warning()
#ifdef __APPLE__
  # if defined(_WIN32)
  # define GLEW_STATIC 1
  # endif
  # include <GL/glew.h>

GLenum err = glewInit(); // defines pters to functions of OpenGL V 1.2 and above
#ifdef FLTK_USE_WAYLAND
  // glewInit returns GLEW_ERROR_NO_GLX_DISPLAY with Wayland
  if (Fl::gl_display() && err == GLEW_ERROR_NO_GLX_DISPLAY) err = GLEW_OK;
#endif
#if (err != GLEW_OK) Fl::warning("glewInit() failed returning \%u", err);
#endif // ! __APPLE__
Changes in the build process

Link with libGLEW.so (with X11 or Wayland), libglew32.a (with MinGW) or glew32.lib (with MS Visual Studio); no change is needed on the Mac OS platform.

1.12 Programming with FLUID

This chapter shows how to use the Fast Light User-Interface Designer ("FLUID") to create your GUIs.

Subchapters:

- What is FLUID?
- Running FLUID Under UNIX
- Running FLUID Under Microsoft Windows
- Compiling .fl Files
- A Short Tutorial
- FLUID Reference
- FLUID Templates
- Internationalization with FLUID
- Known Limitations
- Keyboard Shortcuts
- Licenses

1.12.1 What is FLUID?

The Fast Light User Interface Designer, or FLUID, is a graphical editor that is used to produce FLTK source code. FLUID edits and saves its state in .fl files. These files are text, and you can (with care) edit them in a text editor, perhaps to get some special effects. The .fl file format is explained in detail in fluid/README_fl.txt.

FLUID can "compile" the .fl file into a .cxx and a .h file. The .cxx file defines all the objects from the .fl file and the .h file declares all the global ones. FLUID also supports localization (Internationalization) of label strings and tooltips using message files and the GNU gettext or POSIX catgets interfaces.

A simple program can be made by putting all your code (including a `main()` function) into the .fl file and thus making the .cxx file a single source file to compile. Most programs are more complex than this, so you write other .cxx files that call the FLUID functions. These .cxx files must #include the .h file or they can #include the .cxx file so it still appears to be a single source file.

![Figure 1.28 FLUID organization](image)

Figure 1.28 FLUID organization
Normally the FLUID file defines one or more functions or classes which output C++ code. Each function defines one or more FLTK windows and all the widgets that go inside those windows.

Widgets created by FLUID are either "named", "complex named" or "unnamed". A named widget has a legal C++ variable identifier as its name (i.e. only alphanumeric and underscore). In this case FLUID defines a global variable or class member that will point at the widget after the function defining it is called. A complex named object has punctuation such as ' . ' or '->' or any other symbols in its name. In this case FLUID assigns a pointer to the widget to the name, but does not attempt to declare it. This can be used to get the widgets into structures. An unnamed widget has a blank name and no pointer is stored.

Widgets may either call a named callback function that you write in another source file, or you can supply a small piece of C++ source and FLUID will write a private callback function into the .cxx file.

1.12.2 Running FLUID Under UNIX

To run FLUID under UNIX, type:

```
fluid filename.fl &
```

to edit the .fl file filename.fl. If the file does not exist you will get an error pop-up, but if you dismiss it you will be editing a blank file of that name. You can run FLUID without any name, in which case you will be editing an unnamed blank setup (but you can use save-as to write it to a file).

You can provide any of the standard FLTK switches before the filename:

- `-display hostin.n`
- `-geometry WxH+X+Y`
- `-title windowtitle`
- `-name classname`
- `-iconic`
- `-fg color`
- `-bg color`
- `-bg2 color`
- `-scheme schemename`

Changing the colors may be useful to see what your interface will look at if the user calls it with the same switches. Similarly, using `-scheme plastic` will show how the interface will look using the "plastic" scheme.

In the current version, if you don't put FLUID into the background with ' & ' then you will be able to abort FLUID by typing CTRL-C on the terminal. It will exit immediately, losing any changes.

1.12.3 Running FLUID Under Microsoft Windows

To run FLUID under Windows, double-click on the FLUID.exe file. You can also run FLUID from the Command Prompt window. FLUID always runs in the background under Windows.

1.12.4 Compiling .fl Files

FLUID can also be called as a command-line "compiler" to create the .cxx and .h file from a .fl file. To do this type:

```
fluid -c filename.fl
```

This is the same as the menu 'File/Write Code...'. It will read the filename.fl file and write filename.cxx and filename.h. Any leading directory on filename.fl will be stripped, so they are always written to the current directory. If there are any errors reading or writing the files, FLUID will print the error and exit with a non-zero code. You can use the following lines in a makefile to automate the creation of the source and header files:

```
my_panels.h my_panels.cxx: my_panels.fl
fluid -c my_panels.fl
```
1.12 Programming with FLUID

Most versions of make support rules that cause .fl files to be compiled:

```
.SUFFIXES: .fl .cxx .h
.fl.h .fl.cxx:
  fluid -c $<
```

If you use
```
fluid -cs filename.fl
```
FLUID will also write the "strings" for internationalization in file 'filename.txt' (menu: ‘File/Write Strings...’).

Finally there is another option which is useful for program developers who have many .fl files and want to upgrade them to the current FLUID version. FLUID will read the filename.fl file, save it, and exit immediately. This writes the file with current syntax and options and the current FLTK version in the header of the file. Use
```
fluid -u filename.fl
```
to 'upgrade' filename.fl. You may combine this with '-c' or '-cs'.

Note

All these commands overwrite existing files w/o warning. You should particularly take care when running ‘fluid
-u’ since this overwrites the original .fl source file.

1.12.5 A Short Tutorial

FLUID is an amazingly powerful little program. However, this power comes at a price as it is not always obvious how to accomplish seemingly simple tasks with it. This tutorial will show you how to generate a complete user interface class with FLUID that is used for the CubeView program provided with FLTK.

![Figure 1.29 CubeView demo](image)

The window is of class CubeViewUI, and is completely generated by FLUID, including class member functions. The central display of the cube is a separate subclass of Fl_Gl_Window called CubeView. CubeViewUI manages CubeView using callbacks from the various sliders and rollers to manipulate the viewing angle and zoom of CubeView.

At the completion of this tutorial you will (hopefully) understand how to:
1. Use FLUID to create a complete user interface class, including constructor and any member functions necessary.

2. Use FLUID to set callback member functions of custom widget classes.

3. Subclass an Fl_Gl_Window to suit your purposes.

1.12.5.1 The CubeView Class

The CubeView class is a subclass of Fl_Gl_Window. It has methods for setting the zoom, the x and y pan, and the rotation angle about the x and y axes.

You can safely skip this section as long as you realize that CubeView is a subclass of Fl_Gl_Window and will respond to calls from CubeViewUI, generated by FLUID.

The CubeView Class Definition

Here is the CubeView class definition, as given by its header file "test/CubeView.h":

```cpp
#include <FL/Fl.H>
#include <FL/Fl_Gl_Window.H>
#include <FL/gl.h>

class CubeView : public Fl_Gl_Window {
public:
    CubeView(int x, int y, int w, int h, const char *l = 0);
    // This value determines the scaling factor used to draw the cube.
    double size;
    /* Set the rotation about the vertical (y) axis.
       This function is called by the horizontal roller in
       CubeViewUI and the initialize button in CubeViewUI.
       */
    void v_angle(double angle) { vAng = angle; }
    // Return the rotation about the vertical (y) axis.
    double v_angle() const { return vAng; }
    /* Set the rotation about the horizontal (x) axis.
       This function is called by the vertical roller in
       CubeViewUI and the initialize button in CubeViewUI.
       */
    void h_angle(double angle) { hAng = angle; }
    // The rotation about the horizontal (x) axis.
    double h_angle() const { return hAng; }
    /* Sets the x shift of the cube view camera.
       This function is called by the slider in CubeViewUI
       and the initialize button in CubeViewUI.
       */
    void panx(double x) { xshift = x; }
    /* Sets the y shift of the cube view camera.
       This function is called by the slider in CubeViewUI
       and the initialize button in CubeViewUI.
       */
    void pany(double y) { yshift = y; }
    /* The widget class draw() override.
       The draw() function initializes Gl for another round of
       drawing, then calls specialized functions for drawing each
       of the entities displayed in the cube view.
       */
    void draw();

    // Additional member functions for zoom, pan, and rotation

    // Additional member variables for size, rotation angles, and shifts

};
```

Generated by Doxygen
private:
    /* Draw the cube boundaries. 
     * Draw the faces of the cube using the boxv[] vertices, 
     * using GL_LINE_LOOP for the faces. 
     */
    void drawCube();

    double vAng, hAng;
    double xshift, yshift;

    float boxv0[3]; float boxv1[3];
    float boxv2[3]; float boxv3[3];
    float boxv4[3]; float boxv5[3];
    float boxv6[3]; float boxv7[3];
};

The CubeView Class Implementation

Here is the CubeView implementation. It is very similar to the "CubeView" demo included with FLTK.

#include "CubeView.h"
#include <math.h>

CubeView::CubeView(int x, int y, int w, int h, const char *l)
    : Fl_Gl_Window(x, y, w, h, l)
    { 
    Fl::use_high_res_GL(1);
    vAng = 0.0;
    hAng = 0.0;
    size = 10.0;
    xshift = 0.0;
    yshift = 0.0;

    /* The cube definition. These are the vertices of a unit cube 
     * centered on the origin. */

    boxv0[0] = -0.5; boxv0[1] = -0.5; boxv0[2] = -0.5;
    boxv1[0] = 0.5; boxv1[1] = -0.5; boxv1[2] = -0.5;
    boxv2[0] = 0.5; boxv2[1] = 0.5; boxv2[2] = -0.5;
    boxv3[0] = -0.5; boxv3[1] = 0.5; boxv3[2] = -0.5;
    boxv4[0] = -0.5; boxv4[1] = -0.5; boxv4[2] = 0.5;
    boxv5[0] = 0.5; boxv5[1] = -0.5; boxv5[2] = 0.5;
    boxv6[0] = 0.5; boxv6[1] = 0.5; boxv6[2] = 0.5;
    boxv7[0] = -0.5; boxv7[1] = 0.5; boxv7[2] = 0.5;
}

void CubeView::drawCube() {
    /* Draw a colored cube */
    #define ALPHA 0.5
    glBegin(GL_QUADS);
    glColor4f(0.0, 0.0, 1.0, ALPHA);
    glVertex3fv(boxv0);
    glVertex3fv(boxv1);
    glVertex3fv(boxv2);
    glVertex3fv(boxv3);
    glColor4f(1.0, 1.0, 0.0, ALPHA);
    glVertex3fv(boxv0);
    glVertex3fv(boxv4);
    glVertex3fv(boxv5);
    glVertex3fv(boxv1);
    glColor4f(0.0, 1.0, 1.0, ALPHA);
    glVertex3fv(boxv2);
    glVertex3fv(boxv6);
    glVertex3fv(boxv7);
    glVertex3fv(boxv3);
    glColor4f(1.0, 0.0, 0.0, ALPHA);
    glVertex3fv(boxv4);
    glVertex3fv(boxv5);
    glVertex3fv(boxv6);
    glVertex3fv(boxv7);
    glColor4f(1.0, 0.0, 1.0, ALPHA);
    glVertex3fv(boxv0);
    glEnd();
}
glVertex3fv(boxv3);
glVertex3fv(boxv7);
glVertex3fv(boxv4);

glColor4f(0.0, 1.0, 0.0, ALPHA);
glVertex3fv(boxv1);
glVertex3fv(boxv5);
glVertex3fv(boxv6);
glVertex3fv(boxv2);
glEnd();

glColor3f(1.0, 1.0, 1.0);

void CubeView::draw() {
  if (!valid()) {
    glLoadIdentity();
    glViewport(0, 0, pixel_w(), pixel_h());
    glOrtho(-10, 10, -10, 10, -20050, 10000);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
  }
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glPushMatrix();
  glTranslatef((GLfloat)xshift, (GLfloat)yshift, 0);
  glRotatef((GLfloat)hAng, 0, 1, 0);
  glRotatef((GLfloat)vAng, 1, 0, 0);
  glScalef(float(size), float(size), float(size));
  drawCube();
  glPopMatrix();
}

1.12.5.2 The CubeViewUI Class

We will completely construct a window to display and control the CubeView defined in the previous section using FLUID.
Defining the CubeViewUI Class

Once you have started FLUID, the first step in defining a class is to create a new class within FLUID using the New->Code->Class menu item. Name the class "CubeViewUI" and leave the subclass blank. We do not need any inheritance for this window. You should see the new class declaration in the FLUID browser window.

![FLUID file for CubeView](image)

Figure 1.30 FLUID file for CubeView

Adding the Class Constructor

Click on the CubeViewUI class in the FLUID window and add a new method by selecting New->Code->Function/Method. The name of the function will also be CubeViewUI. FLUID will understand that this will be the constructor for the class and will generate the appropriate code. Make sure you declare the constructor public.

Then add a window to the CubeViewUI class. Highlight the name of the constructor in the FLUID browser window and click on New->Group->Window. In a similar manner add the following to the CubeViewUI constructor:

- A horizontal roller named hrot
- A vertical roller named vrot
- A horizontal slider named xpan
- A vertical slider named ypan
- A horizontal value slider named zoom

None of these additions need be public. And they shouldn't be unless you plan to expose them as part of the interface for CubeViewUI.

When you are finished you should have something like this:
We will talk about the `show()` method that is highlighted shortly.

Adding the CubeView Widget

What we have is nice, but does little to show our cube. We have already defined the CubeView class and we would like to show it within the CubeViewUI.

The CubeView class inherits the `FL_Gl_Window` class, which is created in the same way as an `Fl_Box` widget. Use `New->Other->Box` to add a square box to the main window. This will be no ordinary box, however.

The Box properties window will appear. The key to letting CubeViewUI display CubeView is to enter CubeView in the **Class**: text entry box. This tells FLUID that it is not an `Fl_Box`, but a similar widget with the same constructor.

In the **Extra Code**: field enter `#include "CubeView.h"

This `#include` is important, as we have just included CubeView as a member of CubeViewUI, so any public CubeView methods are now available to CubeViewUI.
Defining the Callbacks

Each of the widgets we defined before adding CubeView can have callbacks that call CubeView methods. You can call an external function or put a short amount of code in the Callback field of the widget panel. For example, the callback for the ypan slider is:

```c
cube->pany(((Fl_Slider *)o)->value());
cube->redraw();
```

We call `cube->redraw()` after changing the value to update the CubeView window. CubeView could easily be modified to do this, but it is nice to keep this exposed. In the case where you may want to do more than one view change only redrawing once saves a lot of time.

There is no reason to wait until after you have added CubeView to enter these callbacks. FLUID assumes you are smart enough not to refer to members or functions that don't exist.

Adding a Class Method

You can add class methods within FLUID that have nothing to do with the GUI. As an example add a show function so that CubeViewUI can actually appear on the screen.

Make sure the top level CubeViewUI is selected and select New->Code->Function/Method. Just use the name `show()`. We don't need a return value here, and since we will not be adding any widgets to this method FLUID will assign it a return type of `void`. 

---

Figure 1.32 CubeView methods
Once the new method has been added, highlight its name and select **New-&gt;Code-&gt;Code**. Enter the method’s code in the code window.

1.12.5.3 Adding Constructor Initialization Code

If you need to add code to initialize a class, for example setting initial values of the horizontal and vertical angles in the CubeView, you can simply highlight the constructor and select **New-&gt;Code-&gt;Code**. Add any required code.

1.12.5.4 Generating the Code

Now that we have completely defined the CubeViewUI, we have to generate the code. There is one last trick to ensure this all works. Open the preferences dialog from **Edit-&gt;Preferences**.

At the bottom of the preferences dialog box is the key: "**Include Header from Code**". Select that option and set your desired file extensions and you are in business. You can include the CubeViewUI.h (or whatever extension you prefer) as you would any other C++ class.

1.12.6 FLUID Reference

The following sections describe each of the windows in FLUID.

1.12.6.1 The Widget Browser

The main window shows a menu bar and a scrolling browser of all the defined widgets. The name of the *.fl* file being edited is shown in the window title.

The widgets are stored in a hierarchy. You can open and close a level by clicking the “triangle” at the left of a widget. The leftmost widgets are the **parents**, and all the widgets listed below them are their **children**. Parents don’t have to have any children.
The top level of the hierarchy is composed of functions and classes. Each of these will produce a single C++ public function or class in the output .cxx file. Calling the function or instantiating the class will create all of the child widgets.

The second level of the hierarchy contains the windows. Each of these produces an instance of class Fl_Window.

Below that are either widgets (subclasses of Fl_Widget) or groups of widgets (including other groups). Plain groups are for layout, navigation, and resize purposes. Tab groups provide the well-known file-card tab interface.

Widgets are shown in the browser by either their name (such as "main_panel" in the example), or by their type and label (such as "Button "the green").

You select widgets by clicking on their names, which highlights them (you can also select widgets from any displayed window). You can select many widgets by dragging the mouse across them, or by using Shift+Click to toggle them on and off. To select no widgets, click in the blank area under the last widget. Note that hidden children may be selected even when there is no visual indication of this.

You open widgets by double-clicking on them, or (to open several widgets you have picked) by typing the F1 key. A control panel will appear so you can change the widget(s).

1.12.6.2 Menu Items

The menu bar at the top is duplicated as a pop-up menu on any displayed window. The shortcuts for all the menu items work in any window. The menu items are:

File/Open... (Ctrl+o)

Discards the current editing session and reads in a different .fl file. You are asked for confirmation if you have changed the current file.

FLUID can also read .fd files produced by the Forms and XForms "fdesign" programs. It is best to File/→Merge them instead of opening them. FLUID does not understand everything in a .fd file, and will print a warning message on the controlling terminal for all data it does not understand. You will probably need to edit the resulting setup to fix these errors. Be careful not to save the file without changing the name, as FLUID will write over the .fd file with its own format, which fdesign cannot read!

File/Insert... (Ctrl+i)

Inserts the contents of another .fl file, without changing the name of the current .fl file. All the functions (even if they have the same names as the current ones) are added, and you will have to use cut/paste to put the widgets where you want.
File/Save (Ctrl+s)

Writes the current data to the .fl file. If the file is unnamed then FLUID will ask for a filename.

File/Save As... (Ctrl+Shift+S)

Asks for a new filename and saves the file.

File/Save A Copy...

Save a copy of the .fl data to a different file.

File/Revert...

Revert the .fl data to a previously saved state.

File/New From Template...

Create a new user interface design from a previously saved template. This can be useful for including a predefined enterprise copyright message for projects, or for managing boilerplate code for repeating project code.

File/Save As Template...

Save the current project as a starting point for future projects.

File/Print... (Ctrl-P)
Generate a printout containing all currently open windows within your project.

**File/Write Code (Ctrl+Shift+C)**

“Compiles” the data into a `.cxx` and `.h` file. These are exactly the same as the files you get when you run FLUID with the `-c` switch.

The output file names are the same as the `.fl` file, with the leading directory and trailing ".fl" stripped, and ".h" or ".cxx" appended.

**File/Write Strings (Ctrl+Shift+W)**

Writes a message file for all of the text labels defined in the current file.

The output file name is the same as the `.fl` file, with the leading directory and trailing ".fl" stripped, and ".txt", ".po", or ".msg" appended depending on the Internationalization Mode.

**File/Quit (Ctrl+q)**

Exits FLUID. You are asked for confirmation if you have changed the current file.

**Edit/Undo (Ctrl+z) and Redo (Shift+Ctrl+z)**

FLUID saves the project state for undo and redo operations after every major change.

**Edit/Cut (Ctrl+x)**
Deletes the selected widgets and all of their children. These are saved to a "clipboard" file and can be pasted back into any FLUID window.

**Edit/Copy (Ctrl+c)**

Copies the selected widgets and all of their children to the "clipboard" file.

**Edit/Paste (Ctrl+v)**

Pastes the widgets from the clipboard file.

If the widget is a window, it is added to whatever function is selected, or contained in the current selection.

If the widget is a normal widget, it is added to whatever window or group is selected. If none is, it is added to the window or group that is the parent of the current selection.

To avoid confusion, it is best to select exactly one widget before doing a paste.

Cut/paste is the only way to change the parent of a widget.

**Edit/Select All (Ctrl+a)**

Selects all widgets in the same group as the current selection.

If they are all selected already then this selects all widgets in that group's parent. Repeatedly typing Ctrl+a will select larger and larger groups of widgets until everything is selected.

**Edit/Properties... (F1 or double click)**
Displays the current widget in the attributes panel. If the widget is a window and it is not visible then the window is shown instead.

Edit/Sort

Sorts the selected widgets into left to right, top to bottom order. You need to do this to make navigation keys in FLTK work correctly. You may then fine-tune the sorting with "Earlier" and "Later". This does not affect the positions of windows or functions.

Edit/Earlier (F2)

Moves all of the selected widgets one earlier in order among the children of their parent (if possible). This will affect navigation order, and if the widgets overlap it will affect how they draw, as the later widget is drawn on top of the earlier one. You can also use this to reorder functions, classes, and windows within functions.

Edit/Later (F3)

Moves all of the selected widgets one later in order among the children of their parent (if possible).

Edit/Group (F7)

Creates a new Fl_Group and make all the currently selected widgets children of it.

Edit/Ungroup (F8)

Deletes the parent group if all the children of a group are selected.

Edit/Show or Hide Overlays (Ctrl+Shift+O)
Toggles the display of the red overlays off, without changing the selection. This makes it easier to see box borders and how the layout looks. The overlays will be forced back on if you change the selection.

**Edit/Show or Hide Guides (Ctrl+Shift+G)**

Guides can be used to arrange a widget layout easily and consistently. They indicating preferred widget positions and sizes with user definable margins, grids, and gap sizes. See the "Layout" tab in the "Settings" dialog.

This menu item enables and disable guides and the snapping action when dragging widgets and their borders.

**Edit/Show or Hide Restricted (Ctrl+Shift+R)**

It is recommended to ensure that user interface elements within FLTK do not exceed the size of their container group or overlap with each other. By activating this button, a hatch pattern is enabled, visually highlighting areas where restricted or undefined behavior may occur.

**Edit/Show or Hide Widget Bin (Alt+B)**

The widget bin provides quick access to all widget types supported by FLUID. Layouts can be created by clicking on elements in the widget bin, or by dragging them from the bin to their position within the layout. This button shows or hides the widget bin.

**Edit/Show or Hide Source Code (Shift+Alt+S)**

This button shows or hides the source code preview window. Any change to the layout or code in the layout editors can be previewed and verified immediately in the Code View window.

**Edit/Settings... (Alt+p)**
Displays the FLUID settings panel. The settings panel contains tabs for general FLUID settings, project setting, layout preferences, custom shell commands, and the internationalisation settings for GNU Gettext and POSIX `catgets`.

Under the "Project" tab you control the extensions or names of the files that are generated by FLUID. If you check the "Include Header from Code" button the code file will include the header file automatically.

Under the "Locale" tab are the internationalization options, described later in this chapter.

![FLUID General Settings Tab](image.png)

Figure 1.34 FLUID General Settings Tab
New/Code/Function

Creates a new C function. You will be asked for a name for the function. This name should be a legal C++ function template, without the return type. You can pass arguments which can be referred to by code you type into the individual widgets.

If the function contains any unnamed windows, it will be declared as returning an \texttt{FL\_Window} pointer. The unnamed window will be returned from it (more than one unnamed window is useless). If the function contains only named windows, it will be declared as returning nothing (\texttt{void}).

It is possible to make the \texttt{.cxx} output be a self-contained program that can be compiled and executed. This is done by deleting the function name so \texttt{main(argc, argv)} is used. The function will call \texttt{show()} on all the windows it creates and then call \texttt{Fl::run().} This can also be used to test resize behavior or other parts of the user interface.

You can change the function name by double-clicking on the function.
Creates a new Fl_Window widget. The window is added to the currently selected function, or to the function containing the currently selected item. The window will appear, sized to 100x100. You can resize it to whatever size you require.

The widget panel will also appear and is described later in this chapter.

New/...

All other items on the New menu are subclasses of Fl_Widget. Creating them will add them to the currently selected group or window, or the group or window containing the currently selected widget. The initial dimensions and position are chosen by copying the current widget, if possible.

When you create the widget you will get the widget's control panel, which is described later in this chapter.

Layout/Align/...

Align all selected widgets to the first widget in the selection.

Layout/Space Evenly/...

Space all selected widgets evenly inside the selected space. Widgets will be sorted from first to last.

Layout/Make Same Size/...

Make all selected widgets the same size as the first selected widget.

Layout/Center in Group/...
Center all selected widgets relative to their parent widget

Layout/Grid and Size Settings... (Ctrl+g)

Displays the grid settings panel.

This panel controls dimensions that all widgets snap to when you move and resize them, and for the "snap" which is how far a widget has to be dragged from its original position to actually change.

Layout preferences are defined using margins to parent groups and window, gaps between widget, and/or by overlaying a grid over a group or window. A layout comes as a suite of three presets, one for the main application window, one for dialog boxes, and one for toolboxes.

FLUID comes with two include layout suites. FLTK was used to design FLUID and other included apps, and Grid is a more rigid grid layout. Users can add more suites, import and export them, and include them into their .fl project files.
1.12 Programming with FLUID

Figure 1.36 FLUID Layout/Grid Settings Window

Shell/Execute Command... (Alt+x)

Displays the shell command panel. The shell command is commonly used to run a 'make' script to compile the FLTK output.
Shell/Execute Again (Alt+g)

Run the shell command again.

Help/Repid development with FLUID

Shows this chapter of the manual.

Help/FLTK Programmers Manual

Shows the contents page of the manual.

Help/About FLUID

Pops up a panel showing the version of FLUID.

1.12.6.3 The Widget Panel

When you double-click on a widget or a set of widgets you will get the "widget attribute panel".

When you change attributes using this panel, the changes are reflected immediately in the window. It is useful to hit the "Hide Overlays" button (or type Ctrl+Shift+O) to hide the red overlay so you can see the widgets more accurately, especially when setting the box type.

If you have several widgets selected, they may have different values for the fields. In this case the value for one of the widgets is shown. But if you change this value, all of the selected widgets are changed to the new value.

Hitting "OK" makes the changes permanent. Selecting a different widget also makes the changes permanent. FLUID checks for simple syntax errors such as mismatched parenthesis in any code before saving any text.
1.12 Programming with FLUID

1.12.7 GUI Attributes

Not all fields in the Widget attributes dialog will be visible for all types of widgets.

Label (text field)

String to print next to or inside the button. You can put newlines into the string to make multiple lines. The easiest way is by typing Ctrl+j.

Symbols can be added to the label using the at sign (@).

Label (pull down menu)

How to draw the label. Normal, shadowed, engraved, and embossed change the appearance of the text.
Image

The active image for the widget. Click on the **Browse...** button to pick an image file using the file chooser.

Inactive

The inactive image for the widget. Click on the **Browse...** button to pick an image file using the file chooser.

Alignment (buttons)

Where to draw the label. The arrows put it on that side of the widget, you can combine them to put it in the corner. The "box" button puts the label inside the widget, rather than outside.

The **clip** button clips the label to the widget box, the **wrap** button wraps any text in the label, and the **text image** button puts the text over the image instead of under the image.

Position (text fields)

The position fields show the current position and size of the widget box. Enter new values to move and/or resize a widget.

These fields understand basic math and variables. Appending +10 to the **X** coordinate will move a widget 10 units to the right without having to reenter the value. Entering the formula **w+3** in the **Width** field will widen all selected Widgets by 3 units. The formula **py+i*20** in the **Y** field will order all selected widgets vertically in their group by increments of 20 units.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>zero based counter of selected widgets</td>
</tr>
<tr>
<td>x, y, w, h</td>
<td>position and size of the current widget</td>
</tr>
<tr>
<td>px, py, pw, ph</td>
<td>dimensions of the parent widget</td>
</tr>
<tr>
<td>sx, sy, sw, sh</td>
<td>dimensions of the previous sibling</td>
</tr>
<tr>
<td>ex, ey, ew, eh</td>
<td>bounding box of all children</td>
</tr>
</tbody>
</table>
Values (text fields)

The values and limits of the current widget. Depending on the type of widget, some or all of these fields may be inactive.

Shortcut

The shortcut key to activate the widget. Click on the shortcut button and press any key sequence to set the shortcut.

Attributes (buttons)

The **Visible** button controls whether the widget is visible (on) or hidden (off) initially. Don't change this for windows or for the immediate children of a Tabs group.

The **Active** button controls whether the widget is activated (on) or deactivated (off) initially. Most widgets appear greyed out when deactivated.

The **Resizable** button controls whether the window is resizeable. In addition all the size changes of a window or group will go "into" the resizable child. If you have a large data display surrounded by buttons, you probably want that data area to be resizable. You can get more complex behavior by making invisible boxes the resizable widget, or by using hierarchies of groups. Unfortunately the only way to test it is to compile the program. Resizing the FLUID window is *not* the same as what will happen in the user program.

The **Hotspot** button causes the parent window to be positioned with that widget centered on the mouse. This position is determined *when the FLUID function is called*, so you should call it immediately before showing the window. If you want the window to hide and then reappear at a new position, you should have your program set the hotspot itself just before `show()`.

The **Border** button turns the window manager border on or off. On most window managers you will have to close the window and reopen it to see the effect.
X Class (text field)

The string typed into here is passed to the X window manager as the class. This can change the icon or window decorations. On most (all?) window managers you will have to close the window and reopen it to see the effect.

![Figure 1.38 The FLUID widget Style attributes](image)

1.12.7.1 Style Attributes

Label Font (pulldown menu)

Font to draw the label in. Ignored by symbols, bitmaps, and pixmaps. Your program can change the actual font used by these “slots” in case you want some font other than the 16 provided.

Label Size (pulldown menu)

Pixel size (height) for the font to draw the label in. Ignored by symbols, bitmaps, and pixmaps. To see the result without dismissing the panel, type the new number and then Tab.
Label Color (button)

Color to draw the label. Ignored by pixmaps (bitmaps, however, do use this color as the foreground color).

Box (pulldown menu)

The boxtype to draw as a background for the widget.

Many widgets will work, and draw faster, with a "frame" instead of a "box". A frame does not draw the colored interior, leaving whatever was already there visible. Be careful, as FLUID may draw this ok but the real program may leave unwanted stuff inside the widget.

If a window is filled with child widgets, you can speed up redrawing by changing the window's box type to "NO_BOX". FLUID will display a checkerboard for any areas that are not colored in by boxes. Note that this checkerboard is not drawn by the resulting program. Instead random garbage will be displayed.

Down Box (pulldown menu)

The boxtype to draw when a button is pressed or for some parts of other widgets like scrollbars and valuators.

Color (button)

The color to draw the box with.

Select Color (button)

Some widgets will use this color for certain parts. FLUID does not always show the result of this: this is the color buttons draw in when pushed down, and the color of input fields when they have the focus.
Text Font, Size, and Color

Some widgets display text, such as input fields, pull-down menus, and browsers.

![FLUID widget C++ attributes](image)

**Figure 1.39 The FLUID widget C++ attributes**

### 1.12.7.2 C++ Attributes

#### Class

This is how you use your own subclasses of `Fl_Widget`. Whatever identifier you type in here will be the class that is instantiated.

In addition, no `#include` header file is put in the `.h` file. You must provide a `#include` line as the first line of the "Extra Code" which declares your subclass.

The class must be similar to the class you are spoofing. It does not have to be a subclass. It is sometimes useful to change this to another FLTK class. For windows you can select either `Single` or `Double` in the drop-down box right to the "Class:" field to get a normal window (`Fl_Window`) or a double-buffered window (`Fl_Double_Window`), respectively.
Type (upper-right pulldown menu)

Some classes have subtypes that modify their appearance or behavior. You pick the subtype off of this menu.

Name (text field)

Name of a variable to declare, and to store a pointer to this widget into. This variable will be of type "<class>∗". If the name is blank then no variable is created.

You can name several widgets with "name[0]", "name[1]", "name[2]", etc. This will cause FLUID to declare an array of pointers. The array is big enough that the highest number found can be stored. All widgets in the array must be the same type.

Public (button)

Controls whether the widget is publicly accessible. When embedding widgets in a C++ class, this controls whether the widget is public or private in the class. Otherwise it controls whether the widget is declared static or global (extern).

Extra Code (text fields)

These four fields let you type in literal lines of code to dump into the .h or .cxx files.

If the text starts with a # or the word extern then FLUID thinks this is an "include" line, and it is written to the .h file. If the same include line occurs several times then only one copy is written.

All other lines are "code" lines. The current widget is pointed to by the local variable o. The window being constructed is pointed to by the local variable w. You can also access any arguments passed to the function here, and any named widgets that are before this one.
FLUID will check for matching parenthesis, braces, and quotes, but does not do much other error checking. Be careful here, as it may be hard to figure out what widget is producing an error in the compiler. If you need more than four lines you probably should call a function in your own .cxx code.

Callback (text field)

This can either be the name of a function, or a small snippet of code. If you enter anything other than letters, numbers, and the underscore then FLUID treats it as code.

A name refers to a function in your own code. It must be declared as `void name(<class>*, void*)`.

A code snippet is inserted into a static function in the .cxx output file. The function prototype is `void name(class *o, void *v)` so that you can refer to the widget as `o` and the `user_data()` as `v`. FLUID will check for matching parenthesis, braces, and quotes, but does not do much other error checking. Be careful here, as it may be hard to figure out what widget is producing an error in the compiler.

If the callback is blank then no callback is set.

User Data (text field)

This is a value for the `user_data()` of the widget. If blank the default value of zero is used. This can be any piece of C code that can be cast to a void pointer.

Type (text field)

The `void*` in the callback function prototypes is replaced with this. You may want to use `long` for old XForms code. Be warned that anything other than `void*` is not guaranteed to work! However on most architectures other pointer types are ok, and `long` is usually ok, too.

When (pulldown menu)

When to do the callback. This can be Never, Changed, Release, or Enter Key. The value of Enter Key is only useful for text input fields.

There are other rare but useful values for the when() field that are not in the menu. You should use the extra code fields to put these values in.

No Change (button)

The No Change button means the callback is done on the matching event even if the data is not changed.
1.12.8 Selecting and Moving Widgets

Double-clicking a window name in the browser will display it, if not displayed yet. From this display you can select widgets, sets of widgets, and move or resize them. To close a window either double-click it or type ESC.

To select a widget, click it. To select several widgets drag a rectangle around them. Holding down shift will toggle the selection of the widgets instead.

You cannot pick hidden widgets. You also cannot choose some widgets if they are completely overlapped by later widgets. Use the browser to select these widgets.

The selected widgets are shown with a red "overlay" line around them. You can move the widgets by dragging this box. Or you can resize them by dragging the outer edges and corners. Hold down the Alt key while dragging the mouse to defeat the snap-to-grid effect for fine positioning.

If there is a tab box displayed you can change which child is visible by clicking on the file tabs. The child you pick is selected.

The arrow, tab, and shift+tab keys "navigate" the selection. Left, right, tab, or shift+tab move to the next or previous widgets in the hierarchy. Hit the right arrow enough and you will select every widget in the window. Up/down widgets move to the previous/next widgets that overlap horizontally. If the navigation does not seem to work you probably need to "Sort" the widgets. This is important if you have input fields, as FLTK uses the same rules when using arrow keys to move between input fields.

To "open" a widget, double click it. To open several widgets select them and then type F1 or pick "Edit/Open" off the pop-up menu.

Type Ctrl+o to temporarily toggle the overlay off without changing the selection, so you can see the widget borders.

You can resize the window by using the window manager border controls. FLTK will attempt to round the window size to the nearest multiple of the grid size and makes it big enough to contain all the widgets (it does this using illegal X methods, so it is possible it will barf with some window managers!). Notice that the actual window in your program may not be resizable, and if it is, the effect on child widgets may be different.

The panel for the window (which you get by double-clicking it) is almost identical to the panel for any other Fl_Widget. There are three extra items:

1.12.9 Image Labels

The contents of the image files in the Image and Inactive text fields are written to the .cxx file. If many widgets share the same image then only one copy is written. Since the image data is embedded in the generated source code, you need only distribute the C++ code and not the image files themselves.

However, the filenames are stored in the .fl file so you will need the image files as well to read the .fl file. Filenames are relative to the location of the .fl file and not necessarily the current directory. We recommend you either put the images in the same directory as the .fl file, or use absolute path names.

Notes for All Image Types

FLUID runs using the default visual of your X server. This may be 8 bits, which will give you dithered images. You may get better results in your actual program by adding the code "Fl::visual(FL_RGB)" to your code right before the first window is displayed.
All widgets with the same image on them share the same code and source X pixmap. Thus once you have put an image on a widget, it is nearly free to put the same image on many other widgets.

If you edit an image at the same time you are using it in FLUID, the only way to convince FLUID to read the image file again is to remove the image from all widgets that are using it or re-load the .fl file.

Don't rely on how FLTK crops images that are outside the widget, as this may change in future versions! The cropping of inside labels will probably be unchanged.

To more accurately place images, make a new "box" widget and put the image in that as the label.

**XBM (X Bitmap) Files**

FLUID reads X bitmap files which use C source code to define a bitmap. Sometimes they are stored with the ".h" or ".bm" extension rather than the standard ".xbm" extension.

FLUID writes code to construct an Fl_Bitmap image and use it to label the widget. The '1' bits in the bitmap are drawn using the label color of the widget. You can change this color in the FLUID widget attributes panel. The '0' bits are transparent.

The program "bitmap" on the X distribution does an adequate job of editing bitmaps.

**XPM (X Pixmap) Files**

FLUID reads X pixmap files as used by the libxpm library. These files use C source code to define a pixmap. The filenames usually have the ".xpm" extension.

FLUID writes code to construct an Fl_Pixmap image and use it to label the widget. The label color of the widget is ignored, even for 2-color images that could be a bitmap. XPM files can mark a single color as being transparent, and FLTK uses this information to generate a transparency mask for the image.
We have not found any good editors for small iconic pictures. For pixmaps we have used XPaint and the KDE icon editor.

**BMP Files**

FLUID reads Windows BMP image files which are often used in Windows applications for icons. FLUID converts BMP files into (modified) XPM format and uses an Fl_BMP_Image image to label the widget. Transparency is handled the same as for XPM files. All image data is uncompressed when written to the source file, so the code may be much bigger than the .bmp file.

**GIF Files**

FLUID reads GIF image files which are often used in HTML documents to make icons. FLUID converts GIF files into (modified) XPM format and uses an Fl_GIF_Image image to label the widget. Transparency is handled the same as for XPM files. All image data is uncompressed when written to the source file, so the code may be much bigger than the .gif file. Only the first image of an animated GIF file is used.

**JPEG Files**

If FLTK is compiled with JPEG support, FLUID can read JPEG image files which are often used for digital photos. FLUID uses an Fl_JPEG_Image image to label the widget, and writes uncompressed RGB or grayscale data to the source file.

**PNG (Portable Network Graphics) Files**

If FLTK is compiled with PNG support, FLUID can read PNG image files which are often used in HTML documents. FLUID uses a Fl_PNG_Image image to label the widget, and writes uncompressed RGB or grayscale data to the source file. PNG images can provide a full alpha channel for partial transparency, and FLTK supports this as best as possible on each platform.
1.12.10 FLUID Templates

Fluid can store a number of project templates. Project templates are great for storing often used boilerplate code for fast access. A common use would be projects with readily prepared copyright messages.

A sample template for FLTK projects is included with Fluid.

Choose "File > New From Template..." to create a new project based on a template file. In the template dialog, select one of the existing templates. All occurrences of the word "@INSTANCE@" in the template are replaced with the text in the "Instance" field. To create the new project click "New".

To add your current project as a new template, choose "File > Save As Template...", fill in a name, and click "Save".

To delete a template, open the template dialog using "New from Template" or "Save As Template", then select any existing template, and click "Delete Template".

1.12.11 Internationalization with FLUID

FLUID supports internationalization (I18N for short) of label strings and tooltips used by widgets. The GNU gettext option also supports deferred translation of statically initialised menu item labels. The preferences window (Ctrl+p) provides access to the I18N options.

1.12.11.1 I18N Methods

FLUID supports three methods of I18N: none, GNU gettext, and POSIX catgets. The "none" method is the default and just passes the label strings as-is to the widget constructors.

The "GNU gettext" method uses GNU gettext (or a similar text-based I18N library) to retrieve a localized string before calling the widget constructor.

The "POSIX catgets" method uses the POSIX catgets function to retrieve a numbered message from a message catalog before calling the widget constructor.
1.12.11.2 Using GNU gettext for I18N

FLUID’s code support for GNU gettext is limited to calling a function or macro to retrieve the localized label; you still need to call setlocale() and textdomain() or bindtextdomain() to select the appropriate language and message file.

To use GNU gettext for I18N, open the preferences window and choose “GNU gettext” from the **Use**: chooser. Four new input fields will then appear to control the include file and function/macro names to use when retrieving localized label strings in dynamic allocation and static initialisation.

![FLUID Settings](image)

**Figure 1.40 Internationalization using GNU gettext**

The **#include** field controls the header file to include for I18N; by default this is `<libintl.h>`, the standard I18N file for GNU gettext.

If the **Conditional**: field contains a macro name, i18n will only be compiled into the product if this macro is defined. The build system should define the macro only if all required headers and libraries are available. If the macro is not defined, no headers are included and _gettext_ passes text through untranslated.

The **Function**: field controls the function (or macro) that will retrieve the localized message; by default the _gettext_ function will be called.

The **Static Function**: field names a macro that will mark static text fields for extraction with the _xgettext_ tool. The default macro name is _gettext_noop_ and will be defined as `#define gettext_noop(text) text` right after the **#include** statement. Fluid will do its best to call _gettext_ on static texts later, after the textdomain was set by the user.

See also

- GNU gettext special cases

1.12.11.3 Using POSIX catgets for I18N

FLUID’s code support for POSIX catgets allows you to use a global message file for all interfaces or a file specific to each .fl file; you still need to call setlocale() to select the appropriate language.
To use POSIX catgets for I18N, open the preferences window and choose "POSIX catgets" from the Use: chooser. Three new input fields will then appear to control the include file, catalog file, and set number for retrieving the localized label strings.

![FLUID Settings](image)

**Figure 1.41 Internationalization using POSIX catgets**

The `#include` field controls the header file to include for I18N; by default this is `<nl_types.h>`, the standard I18N file for POSIX catgets.

The **File:** field controls the name of the catalog file variable to use when retrieving localized messages; by default the file field is empty which forces a local (static) catalog file to be used for all of the windows defined in your `.fl` file.

The **Set:** field controls the set number in the catalog file. The default set is 1 and rarely needs to be changed.

### 1.12.12 Known Limitations

Declaration Blocks can be used to temporarily block out already designed code using `#if 0` and `#endif` type construction. This will effectively avoid compilation of blocks of code. However, static code and data generated by this segment (menu items, images, include statements, etc.) will still be generated and likely cause compile-time warnings.

### 1.12.13 Keyboard Shortcuts

On Apple computers, use the Apple Command key instead of Ctrl.

<table>
<thead>
<tr>
<th>Key Combo</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>widget properties dialog</td>
</tr>
<tr>
<td>F2</td>
<td>move widget earlier in tree</td>
</tr>
<tr>
<td>F3</td>
<td>move widget later in tree</td>
</tr>
<tr>
<td>F7</td>
<td>group widgets</td>
</tr>
<tr>
<td>F8</td>
<td>ungroup widgets</td>
</tr>
<tr>
<td>Delete</td>
<td>delete selected widgets</td>
</tr>
<tr>
<td>Ctrl-1..9</td>
<td>load project from history</td>
</tr>
</tbody>
</table>
### Key Combo | Function
--- | ---
Alt-1 | label text tiny
Alt-2 | label text small
Alt-3 | label text normal
Alt-4 | label text medium
Alt-5 | label text large
Alt-6 | label text huge
Ctrl-A | select all
Shift-Ctrl-A | select none
Alt-B | widget bin
Ctrl-C | copy widgets
Shift-Ctrl-C | generate C code
Ctrl-G | grid setting dialog
Alt-G | execute again
Shift-Alt-G | FLTK settings dialog
Ctrl-I | insert design into project
Ctrl-N | new design
Shift-Ctrl-N | new design from template
Ctrl-O | open design file
Shift-Ctrl-O | toggle overlays
Ctrl-P | print all windows
Alt-P | project settings
Shift-Alt-P | Fluid settings dialog
Ctrl-Q | quit Fluid
Ctrl-S | save design
Shift-Ctrl-S | save design with new name
Shift-Alt-S | source view window
Ctrl-U | duplicate widgets
Ctrl-V | paste widgets
Shift-Ctrl-W | write text strings
Ctrl-X | cut selected Widgets
Alt-X | show 'execute command' dialog
Ctrl-Z | undo
Shift-Ctrl-Z | redo

### Key Combo | Function in interactive Window
--- | ---
LMB | select one widget
Shift-LMB | extend widget selection
Shift-LMB-Drag | window only: resize proportional
Tab | select next
Shift-Tab | select previous
Arrow | move by one unit
Shift-Arrow | resize by one unit
Ctrl-Arrow | move by grid units
Shift-Ctrl-Arrow | resize by grid units

### 1.12.14 Licenses

FLUID uses graphical images based on the Zendesk Garden Stroke icon set: [https://github.com/zendeskgarden](https://github.com/zendeskgarden). Garden Stroke is licensed under the Apache License, Version 2.0: [https://www.apache.org/licenses/LICENSE-2.0.html](https://www.apache.org/licenses/LICENSE-2.0.html).
1.13 FLTK Runtime Options

In this chapter, we will cover how to access and alter settings for applications created using FLTK, both as an administrator and as a regular user.

Subchapters:

- Runtime Options
- Obtaining Current Settings
- Administrative Tool
- List of Options

1.13.1 Runtime Options

FLTK keeps track of various aspects of the user interface in a system-wide database. Users have the ability to set their own preferences and override default or system settings. For instance, FLTK will display a dotted rectangle around the widget with current focus. This might not be desirable for users who do not use keyboard navigation and do not need the rectangle. This can be turned off by setting the OPTION_VISIBLE_FOCUS option to 'off' for that user, which will disable the focus rectangle in all FLTK-based applications.

1.13.2 Obtaining Current Settings

Options are kept in preference files using the signature `Fl_Preferences::CORE_SYSTEM, "fltk.org", "fltk"` for system-wide settings and `Fl_Preferences::CORE_USER, "fltk.org", "fltk"` for individual users. They can be accessed by using the function `bool Fl::option(Fl_Option opt)`. If an application needs to temporarily override user or system settings, it can use the function `void option(Fl_Option opt, bool val)`.

To make changes to options permanently, FLTK provides an administrative tool called fltk-options.

1.13.3 Administrative Tool

`fltk-options` is a hybrid app that is part of FLTK and can be installed on the target system. It includes an up-to-date man page.
When fltk-options is called without any command-line arguments, it opens in interactive mode and provides a user interface to view and alter all system and current user options.

Starting the tool from a shell, the command-line options \texttt{-S} and \texttt{-U} can be used to display or change system or user options. On MS-Windows, \texttt{fltk-options} is also available as \texttt{fltk-options-cmd.exe}.

### 1.13.4 List of Options

Calling \texttt{fltk-options --help} gives a list of all available commands, and options and their values. \texttt{fltk-options --help OPTION} prints more detailed information for \texttt{OPTION} if available. In interactive mode, tooltips provide this additional information.

A full list of options can be found in the manual at \texttt{Fl::Fl_Option}.

### 1.14 Advanced FLTK

This chapter explains advanced programming and design topics that will help you to get the most out of FLTK.

#### 1.14.1 Multithreading

FLTK can be used to implement a GUI for a multithreaded application but, as with multithreaded programming generally, there are some concepts and caveats that must be kept in mind.

Key amongst these is that, for many of the target platforms on which FLTK is supported, only the \texttt{main()} thread of the process is permitted to handle system events, create or destroy windows and open or close windows. Further, only the \texttt{main()} thread of the process can safely write to the display.

To support this in a portable way, all FLTK \texttt{draw()} methods are executed in the \texttt{main()} thread. A worker thread may update the state of an existing widget, but it may not do any rendering directly, nor create or destroy a window.\footnote{A special case exists for \texttt{Fl_Gl_Window} where it can, with suitable precautions, be possible to safely render to an existing GL context from a worker thread.}
Creating portable threads

We do not provide a threading interface as part of the library. A simple example showing how threads can be implemented, for all supported platforms, can be found in `test/threads.h` and `test/threads.cxx`.

FLTK has been used with a variety of thread interfaces, so if the simple example shown in `test/threads.cxx` does not cover your needs, you might want to select a third-party library that provides the features you require.

1.14.2 FLTK multithread locking - Fl::lock() and Fl::unlock()

In a multithreaded program, drawing of widgets (in the `main()` thread) happens asynchronously to widgets being updated by worker threads, so no drawing can occur safely whilst a widget is being modified (and no widget should be modified whilst drawing is in progress).

FLTK supports multithreaded applications using a locking mechanism internally. This allows a worker thread to lock the rendering context, preventing any drawing from taking place, whilst it changes the value of its widget.

Note

The converse is also true; whilst a worker thread holds the lock, the `main()` thread may not be able to process any drawing requests, nor service any events. So a worker thread that holds the FLTK lock must contrive to do so for the shortest time possible or it could impair operation of the application.

The lock operates broadly as follows.

Using the FLTK library, the `main()` thread holds the lock whenever it is processing events or redrawing the display. It acquires (locks) and releases (unlocks) the FLTK lock automatically and no "user intervention" is required. Indeed, a function that runs in the context of the `main()` thread ideally should not acquire / release the FLTK lock explicitly. (Though note that the lock calls are recursive, so calling `Fl::lock()` from a thread that already holds the lock, including the `main()` thread, is benign. The only constraint is that every call to `Fl::lock()` must be balanced by a corresponding call to `Fl::unlock()` to ensure the lock count is preserved.)

The `main()` thread must call `Fl::lock()` once before any windows are shown, to enable the internal lock (it is "off" by default since it is not useful in single-threaded applications) but thereafter the `main()` thread lock is managed by the library internally.

A worker thread, when it wants to alter the value of a widget, can acquire the lock using `Fl::lock()`, update the widget, then release the lock using `Fl::unlock()`. Acquiring the lock ensures that the worker thread can update the widget, without any risk that the `main()` thread will attempt to redraw the widget whilst it is being updated.

Note that acquiring the lock is a blocking action; the worker thread will stall for as long as it takes to acquire the lock. If the `main()` thread is engaged in some complex drawing operation this may block the worker thread for a long time, effectively serializing what ought to be parallel operations. (This frequently comes as a surprise to coders less familiar with multithreaded programming issues; see the discussion of "lockless programming" later for strategies for managing this.)

To incorporate the locking mechanism in the library, FLTK must be compiled with `--enable-threads` set during the `configure` process. IDE-based versions of FLTK are automatically compiled with the locking mechanism incorporated if possible. Since version 1.3, the `configure` script that builds the FLTK library also sets `--enable-threads` by default.
1.14 Advanced FLTK

1.14.3 Simple multithreaded examples using Fl::lock

In `main()`, call `Fl::lock()` once before `Fl::run()` or `Fl::wait()` to enable the lock and start the runtime multithreading support for your program. All callbacks and derived functions like `handle()` and `draw()` will now be properly locked.

This might look something like this:

```c
int main(int argc, char **argv) {
    /* Create your windows and widgets here */
    Fl::lock(); /* "start" the FLTK lock mechanism */
    /* show your window */
    main_win->show(argc, argv);
    /* start your worker threads */
    ... start threads ...
    /* Run the FLTK main loop */
    int result = Fl::run();
    /* terminate any pending worker threads */
    ... stop threads ...
    return result;
}
```

You can start as many threads as you like. From within a thread (other than the `main()` thread) FLTK calls must be wrapped with calls to `Fl::lock()` and `Fl::unlock()`:

```c
void my_thread(void) {
    while (thread_still_running) {
        /* do thread work */
        /* compute new values for widgets */
        ...
        Fl::lock(); // acquire the lock
        my_widget->update(values);
        Fl::unlock(); // release the lock; allow other threads to access FLTK again
        Fl::awake(); // use Fl::awake() to signal main thread to refresh the GUI
    }
}
```

Note

To trigger a refresh of the GUI from a worker thread, the worker code should call `Fl::awake()`

Using `Fl::awake` thread messages

You can send messages from worker threads to the `main()` thread using `Fl::awake(void* message)`. If using this thread message interface, your `main()` might look like this:

```c
int main(int argc, char **argv) {
    /* Create your windows and widgets here */
    Fl::lock(); /* "start" the FLTK lock mechanism */
    /* show your window */
    main_win->show(argc, argv);
    /* start your worker threads */
    ... start threads ...
    /* Run the FLTK loop and process thread messages */
    while (Fl::wait() > 0) {
        if ((next_message = Fl::thread_message()) != NULL) {
            /* process your data, update widgets, etc. */
            ...
        }
    }
    /* terminate any pending worker threads */
    ... stop threads ...
    return 0;
}
```

Your worker threads can send messages to the `main()` thread using `Fl::awake(void* message)`:

```c
void *msg; // "msg" is a pointer to your message
Fl::awake(msg); // send "msg" to main thread
```

A message can be anything you like. The `main()` thread can retrieve the message by calling `Fl::thread_message()`.
**Using Fl::awake callback messages**

You can also request that the `main()` thread call a function on behalf of the worker thread by using `Fl::awake(Fl_Awake_Handler cb, void* userdata)`.

The `main()` thread will execute the callback "as soon as possible" when next processing the pending events. This can be used by a worker thread to perform operations (for example showing or hiding windows) that are prohibited in a worker thread.

```c
void do_something_cb(void *userdata) {
    // Will run in the context of the main thread
    ... do_stuff ...
}
```

```c
// running in worker thread
void *data; // "data" is a pointer to your user data
Fl::awake(do_something_cb, data); // call to execute cb in main thread
```

**Note**

The `main()` thread will execute the Fl_Awake_Handler callback `do_something_cb` asynchronously to the worker thread, at some short but indeterminate time after the worker thread registers the request. When it executes the Fl_Awake_Handler callback, the `main()` thread will use the contents of `*userdata` at the time of execution, not necessarily the contents that `*userdata` had at the time that the worker thread posted the callback request. The worker thread should therefore contrive not to alter the contents of `*userdata` once it posts the callback, since the worker thread does not know when the `main()` thread will consume that data. It is often useful that `userdata` point to a struct, one member of which the `main()` thread can modify to indicate that it has consumed the data, thereby allowing the worker thread to re-use or update `userdata`.

**Warning**

The mechanisms used to deliver `Fl::awake(void* message)` and `Fl::awake(Fl_Awake_Handler cb, void* userdata)` events to the `main()` thread can interact in unexpected ways on some platforms. Therefore, for reliable operation, it is advised that a program use either `Fl::awake(Fl_Awake_Handler cb, void* userdata)` or `Fl::awake(void* message)`, but that they never be intermixed. Calling `Fl::awake()` with no parameters should be safe in either case.

If you have to choose between using the `Fl::awake(void* message)` and `Fl::awake(Fl_Awake_Handler cb, void* userdata)` mechanisms and don't know which to choose, then try the `Fl::awake(Fl_Awake_Handler cb, void* userdata)` method first as it tends to be more powerful in general.

### 1.14.4 FLTK multithreaded "lockless programming"

The simple multithreaded examples shown above, using the FLTK lock, work well for many cases where multiple threads are required. However, when that model is extended to more complex programs, it often produces results that the developer did not anticipate.

A typical case might go something like this. A developer creates a program to process a huge data set. The program has a `main()` thread and 7 worker threads and is targeted to run on an 8-core computer. When it runs, the program divides the data between the 7 worker threads, and as they process their share of the data, each thread updates its portion of the GUI with the results, locking and unlocking as they do so.

But when this program runs, it is much slower than expected and the developer finds that only one of the eight CPU cores seems to be utilised, despite there being 8 threads in the program. What happened?
The threads in the program all run as expected, but they end up being serialized (that is, not able to run in parallel) because they all depend on the single FLTK lock. Acquiring (and releasing) that lock has an associated cost, and is a blocking action if the lock is already held by any other worker thread or by the main() thread.

If the worker threads are acquiring the lock "too often", then the lock will always be held somewhere and every attempt by any other thread (even main()) to lock will cause that other thread (including main()) to block. And blocking main() also blocks event handling, display refresh...

As a result, only one thread will be running at any given time, and the multithreaded program is effectively reduced to being a (complicated and somewhat less efficient) single thread program.

A "solution" is for the worker threads to lock "less often", such that they do not block each other or the main() thread. But judging what constitutes locking "too often" for any given configuration, and hence will block, is a very tricky question. What works well on one machine, with a given graphics card and CPU configuration may behave very differently on another target machine.

There are "interesting" variations on this theme, too: for example it is possible that a "faulty" multithreaded program such as described above will work adequately on a single-core machine (where all threads are inherently serialized anyway and so are less likely to block each other) but then stall or even deadlock in unexpected ways on a multicore machine when the threads do interfere with each other. (I have seen this - it really happens.)

The "better" solution is to avoid using the FLTK lock so far as possible. Instead, the code should be designed so that the worker threads do not update the GUI themselves and therefore never need to acquire the FLTK lock. This would be FLTK multithreaded "lockless programming".

There are a number of ways this can be achieved (or at least approximated) in practice but the most direct approach is for the worker threads to make use of the Fl::awake(Fl_Awake_Handler cb, void* userdata) method so that GUI updates can all run in the context of the main() thread, alleviating the need for the worker thread to ever lock. The onus is then on the worker threads to manage the userdata so that it is delivered safely to the main() thread, but there are many ways that can be done.

Note
Using Fl::awake is not, strictly speaking, entirely "lockless" since the awake handler mechanism incorporates resource locking internally to protect the queue of pending awake messages. These resource locks are held transiently and generally do not trigger the pathological blocking issues described here.

However, aside from using Fl::awake, there are many other ways that a "lockless" design can be implemented, including message passing, various forms of IPC, etc.

If you need high performing multithreaded programming, then take some time to study the options and understand the advantages and disadvantages of each; we can't even begin to scratch the surface of this huge topic here!

And of course occasional, sparse, use of the FLTK lock from worker threads will do no harm; it is "excessive" locking (whatever that might be) that triggers the failing behaviour.

It is always a Good Idea to update the GUI at the lowest rate that is acceptable when processing bulk data (or indeed, in all cases!) Updating at a few frames per second is probably adequate for providing feedback during a long calculation. At the upper limit, anything faster than the frame rate of your monitor and the updates will never even be displayed; why waste CPU computing pixels that you will never show?
1.14.5 FLTK multithreaded Constraints

FLTK supports multiple platforms, some of which allow only the \texttt{main()} thread to handle system events and open or close windows. The safe thing to do is to adhere to the following rules for threads on all operating systems:

- Don't \texttt{show()} or \texttt{hide()} anything that contains \texttt{Fl_Window} based widgets from a worker thread. This includes any windows, dialogs, file choosers, subwindows or widgets using \texttt{Fl_Gl_Window}. Note that this constraint also applies to non-window widgets that have tooltips, since the tooltip will contain a \texttt{Fl_Window} object. The safe and portable approach is \textbf{never} to call \texttt{show()} or \texttt{hide()} on any widget from the context of a worker thread. Instead you can use the \texttt{Fl_Awake_Handler} variant of \texttt{Fl::awake()} to request the \texttt{main()} thread to create, destroy, show or hide the widget on behalf of the worker thread.

- Don't call \texttt{Fl::run()}, \texttt{Fl::wait()}, \texttt{Fl::flush()}, \texttt{Fl::check()} or any related methods that will handle system messages from a worker thread

- Don't intermix use of \texttt{Fl::awake(Fl_Awake_Handler cb, void* userdata)} and \texttt{Fl::awake(void* message)} calls in the same program as they may interact unpredictably on some platforms; choose one or other style of \texttt{Fl::awake(<thing>)} mechanism and use that. (Intermixing calls to \texttt{Fl::awake()} should be safe with either however.)

- Starting with FLTK 1.4, it's possible to start (or cancel) a timer from a worker thread under the condition that the call to \texttt{Fl::add_timeout} (or \texttt{Fl::remove_timeout}) is wrapped in \texttt{Fl::lock()} and \texttt{Fl::unlock()}.

- Don't change window decorations or titles from a worker thread

- The \texttt{make_current()} method will probably not work well for regular windows, but should always work for a \texttt{Fl_Gl_Window} to allow for high speed rendering on graphics cards with multiple pipelines. Managing thread-safe access to the GL pipelines is left as an exercise for the reader! (And may be target specific...)

See also: \texttt{Fl::lock()}, \texttt{Fl::unlock()}, \texttt{Fl::awake()}, \texttt{Fl::awake(Fl_Awake_Handler cb, void* userdata)}, \texttt{Fl::awake(void* message)}, \texttt{Fl::thread_message()}.

1.15 Unicode and UTF-8 Support

This chapter explains how FLTK handles international text via Unicode and UTF-8.

Unicode support was added to FLTK starting with version 1.3.0 and is still incomplete but mostly functional. This chapter is Work in Progress, reflecting the current state of Unicode support.
1.15.1 About Unicode, ISO 10646 and UTF-8

The summary of Unicode, ISO 10646 and UTF-8 given below is deliberately brief and provides just enough information for the rest of this chapter.

For further information, please see:

- https://unicode.org
- https://iso.org
- https://www.cl.cam.ac.uk/~mgk25/unicode.html

The Unicode Standard

The Unicode Standard was originally developed by a consortium of mainly US computer manufacturers and developers of multi-lingual software. It has now become a de-facto standard for character encoding and is supported by most of the major computing companies in the world.

Before Unicode, many different systems, on different platforms, had been developed for encoding characters for different languages, but no single encoding could satisfy all languages. Unicode provides access to over 130,000 characters used in all the major languages written today, and is independent of platform and language.

Unicode also provides higher-level concepts needed for text processing and typographic publishing systems, such as algorithms for sorting and comparing text, composite character and text rendering, right-to-left and bi-directional text handling.

Note

There are currently no plans to add this extra functionality to FLTK.

ISO 10646

The International Organisation for Standardization (ISO) had also been trying to develop a single unified character set. Although both ISO and the Unicode Consortium continue to publish their own standards, they have agreed to coordinate their work so that specific versions of the Unicode and ISO 10646 standards are compatible with each other.

The international standard ISO 10646 defines the Universal Character Set (UCS) which contains the characters required for almost all known languages. The standard also defines three different implementation levels specifying how these characters can be combined.
Note

There are currently no plans for handling the different implementation levels or the combining characters in FLTK.

In UCS, characters have a unique numerical code and an official name, and are usually shown using 'U+' and the code in hexadecimal, e.g. U+0041 is the "Latin capital letter A". The UCS characters U+0000 to U+007F correspond to US-ASCII, and U+0000 to U+00FF correspond to ISO 8859-1 (Latin1).

ISO 10646 was originally designed to handle a 31-bit character set from U+00000000 to U+7FFFFFFF, but the current idea is that 21 bits will be sufficient for all future needs, giving characters up to U+10FFFF. The complete character set is sub-divided into planes. Plane 0, also known as the Basic Multilingual Plane (BMP), ranges from U+0000 to U+FFFFD and consists of the most commonly used characters from previous encoding standards. Other planes contain characters for specialist applications.

Todo  FLTK 1.3 and later supports the full Unicode range (21 bits), but there are a few exceptions, for instance binary shortcut values in menus (Fl_Shortcut) can only be used with characters from the BMP (16 bits). This may be extended in a future FLTK version.

The UCS also defines various methods of encoding characters as a sequence of bytes. UCS-2 encodes Unicode characters into two bytes, which is wasteful if you are only dealing with ASCII or Latin1 text, and insufficient if you need characters above U+00FFFF. UCS-4 uses four bytes, which lets it handle higher characters, but this is even more wasteful for ASCII or Latin1.

UTF-8

The Unicode standard defines various UCS Transformation Formats (UTF). UTF-16 and UTF-32 are based on units of two and four bytes. UCS characters requiring more than 16 bits are encoded using "surrogate pairs" in UTF-16.

UTF-8 encodes all Unicode characters into variable length sequences of bytes. Unicode characters in the 7-bit ASCII range map to the same value and are represented as a single byte, making the transformation to Unicode quick and easy.

All UCS characters above U+007F are encoded as a sequence of several bytes. The top bits of the first byte are set to show the length of the byte sequence, and subsequent bytes are always in the range 0x80 to 0xBF. This combination provides some level of synchronisation and error detection.
Note

This table contains theoretical values outside the valid Unicode range (U+000000 - U+10FFFF). Such values can only be returned by conversion functions for illegal input values (see Illegal Unicode and UTF-8 Sequences).

Moving from ASCII encoding to Unicode will allow all new FLTK applications to be easily internationalized and used all over the world. By choosing UTF-8 encoding, FLTK remains largely source-code compatible to previous iterations of the library.

1.15.2 Unicode in FLTK

Todo Work through the code and this documentation to harmonize the [OksiD] and [fltk2] functions.

FLTK will be entirely converted to Unicode using UTF-8 encoding. If a different encoding is required by the underlying operating system, FLTK will convert the string as needed.

It is important to note that the initial implementation of Unicode and UTF-8 in FLTK involves three important areas:

- provision of Unicode character tables and some simple related functions;
- conversion of char* variables and function parameters from single byte per character representation to UTF-8 variable length sequences;
- modifications to the display font interface to accept general Unicode character or UCS code numbers instead of just ASCII or Latin1 characters.

The current implementation of Unicode / UTF-8 in FLTK will impose the following limitations:

- An implementation note in the [OksiD] code says that all functions are LIMITED to 24 bit Unicode values, but also says that only 16 bits are really used under linux and win32. [Can we verify this?]
- The [fltk2] fl_utf8encode() and fl_utf8decode() functions are designed to handle Unicode characters in the range U+000000 to U+10FFFF inclusive, which covers all UTF-16 characters, as specified in RFC 3629. Note that the user must first convert UTF-16 surrogate pairs to UCS.
- FLTK will only handle single characters, so composed characters consisting of a base character and floating accent characters will be treated as multiple characters.
- FLTK will only compare or sort strings on a byte by byte basis and not on a general Unicode character basis.
- FLTK will not handle right-to-left or bi-directional text.

Todo Verify 16/24 bit Unicode limit for different character sets? OksiD's code appears limited to 16-bit whereas the FLTK2 code appears to handle a wider set. What about illegal characters? See comments in fl_utf8fromwc() and fl_utf8toUtf16().
1.15.3 Illegal Unicode and UTF-8 Sequences

Three pre-processor variables are defined in the source code [1] that determine how fl_utf8decode() handles illegal UTF-8 sequences:

- if ERRORS_TO_CP1252 is set to 1 (the default), fl_utf8decode() will assume that a byte sequence starting with a byte in the range 0x80 to 0x9f represents a Microsoft CP1252 character, and will return the value of an equivalent UCS character. Otherwise, it will be processed as an illegal byte value as described below.

- if STRICT_RFC3629 is set to 1 (not the default!) then UTF-8 sequences that correspond to illegal UCS values are treated as errors. Illegal UCS values include those above U+10FFFF, or corresponding to UTF-16 surrogate pairs. Illegal byte values are handled as described below.

- if ERRORS_TO_ISO8859_1 is set to 1 (the default), the illegal byte value is returned unchanged, otherwise 0xFFFD, the Unicode REPLACEMENT CHARACTER, is returned instead.

[1] Since FLTK 1.3.4 you may set these three pre-processor variables on your compile command line with -D"variable=value" (value: 0 or 1) to avoid editing the source code.

fl_utf8encode() is less strict, and only generates the UTF-8 sequence for 0xFFFD, the Unicode REPLACEMENT CHARACTER, if it is asked to encode a UCS value above U+10FFFF.

Many of the [fltk2] functions below use fl_utf8decode() and fl_utf8encode() in their own implementation, and are therefore somewhat protected from bad UTF-8 sequences.

The [OksiD] fl_utf8len() function assumes that the byte it is passed is the first byte in a UTF-8 sequence, and returns the length of the sequence. Trailing bytes in a UTF-8 sequence will return -1.

- **WARNING:** fl_utf8len() can not distinguish between single bytes representing Microsoft CP1252 characters 0x80-0x9f and those forming part of a valid UTF-8 sequence. You are strongly advised not to use fl_utf8len() in your own code unless you know that the byte sequence contains only valid UTF-8 sequences.

- **WARNING:** Some of the [OksiD] functions below still use fl_utf8len() in their implementations. These may need further validation.

Please see the individual function description for further details about error handling and return values.

1.15.4 FLTK Unicode and UTF-8 Functions

This section provides a brief overview of the functions. For more details, consult the main text for each function via its link.

```
int fl_utf8locale() FLTK2

fl_utf8locale() returns true if the "locale" seems to indicate that UTF-8 encoding is used.

It is highly recommended that you change your system so this does return true!

int fl_utf8test(const char *src, unsigned len) FLTK2
```

Generated by Doxygen
1.15 Unicode and UTF-8 Support

`fl_utf8test()` examines the first `len` bytes of `src`. It returns 0 if there are any illegal UTF-8 sequences; 1 if `src` contains plain ASCII or if `len` is zero; or 2, 3 or 4 to indicate the range of Unicode characters found.

```c
int fl_utf_nb_char(const unsigned char *buf, int len) OksiD
```

Returns the number of UTF-8 characters in the first `len` bytes of `buf`.

```c
int fl_unichar_to_utf8_size(Fl_Unichar)
int fl_utf8bytes(unsigned ucs)
```

Returns the number of bytes needed to encode `ucs` in UTF-8.

```c
int fl_utf8len(char c) OksiD
```

If `c` is a valid first byte of a UTF-8 encoded character sequence, `fl_utf8len()` will return the number of bytes in that sequence. It returns -1 if `c` is not a valid first byte.

```c
unsigned int fl_nonspacing(unsigned int ucs) OksiD
```

Returns true if `ucs` is a non-spacing character.

```c
const char* fl_utf8back(const char *p, const char *start, const char *end) FLTK2
const char* fl_utf8fwd(const char *p, const char *start, const char *end) FLTK2
```

If `p` already points to the start of a UTF-8 character sequence, these functions will return `p`. Otherwise `fl_utf8back()` searches backwards from `p` and `fl_utf8fwd()` searches forwards from `p`, within the `start` and `end` limits, looking for the start of a UTF-8 character.

```c
unsigned int fl_utf8decode(const char *p, const char *end, int *len) FLTK2
int fl_utf8encode(unsigned ucs, char *buf) FLTK2
```

`fl_utf8decode()` attempts to decode the UTF-8 character that starts at `p` and may not extend past `end`. It returns the Unicode value, and the length of the UTF-8 character sequence is returned via the `len` argument. `fl_utf8encode()` writes the UTF-8 encoding of `ucs` into `buf` and returns the number of bytes in the sequence. See the main documentation for the treatment of illegal Unicode and UTF-8 sequences.

```c
unsigned int fl_utf8froma(char *dst, unsigned dstlen, const char *src, unsigned srclen) FLTK2
unsigned int fl_utf8toa(const char *src, unsigned srclen, char *dst, unsigned dstlen) FLTK2
```

Generated by Doxygen
fl_utf8froma() converts a character string containing single bytes per character (i.e. ASCII or ISO-8859-1) into UTF-8. If the src string contains only ASCII characters, the return value will be the same as srclen.

fl_utf8toa() converts a string containing UTF-8 characters into single byte characters. UTF-8 characters that do not correspond to ASCII or ISO-8859-1 characters below 0xFF are replaced with '?'.

Both functions return the number of bytes that would be written, not counting the null terminator. dstlen provides a means of limiting the number of bytes written, so setting dstlen to zero is a means of measuring how much storage would be needed before doing the real conversion.

char* fl_utf2mbcs(const char *src) OksiD

converts a UTF-8 string to a local multi-byte character string. [More info required here!]

unsigned int fl_utf8fromwc(char *dst, unsigned dstlen, const wchar_t *src, unsigned srclen) FLTK2
unsigned int fl_utf8towc(const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen) FLTK2
unsigned int fl_utf8toUtf16(const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen) FLTK2

These routines convert between UTF-8 and wchar_t or "wide character" strings. The difficulty lies in the fact that sizeof(wchar_t) is 2 on Windows and 4 on Linux and most other systems. Therefore some "wide characters" on Windows may be represented as "surrogate pairs" of more than one wchar_t.

fl_utf8fromwc() converts from a "wide character" string to UTF-8. Note that srclen is the number of wchar_t elements in the source string and on Windows this might be larger than the number of characters. dstlen specifies the maximum number of bytes to copy, including the null terminator.

fl_utf8towc() converts a UTF-8 string into a "wide character" string. Note that on Windows, some "wide characters" might result in "surrogate pairs" and therefore the return value might be more than the number of characters. dstlen specifies the maximum number of wchar_t elements to copy, including a zero terminating element. [Is this all worded correctly?]

fl_utf8toUtf16() converts a UTF-8 string into a "wide character" string using UTF-16 encoding to handle the "surrogate pairs" on Windows. dstlen specifies the maximum number of wchar_t elements to copy, including a zero terminating element. [Is this all worded correctly?]
These routines all return the number of elements that would be required for a full conversion of the \texttt{src} string, including the zero terminator. Therefore setting \texttt{dstlen} to zero is a way of measuring how much storage would be needed before doing the real conversion.

\begin{verbatim}
unsigned int fl_utf8from_mb(char *dst, unsigned dstlen, const char *src, unsigned srclen) FLTK2
unsigned int fl_utf8to_mb(const char *src, unsigned srclen, char *dst, unsigned dstlen) FLTK2
\end{verbatim}

These functions convert between UTF-8 and the locale-specific multi-byte encodings used on some systems for filenames, etc. If \texttt{fl_utf8locale()} returns true, these functions don't do anything useful. [Is this all worded correctly?]

\begin{verbatim}
int fl_tolower(unsigned int ucs) OksiD
int fl_toupper(unsigned int ucs) OksiD
int fl_utf_tolower(const unsigned char *str, int len, char *buf) OksiD
int fl_utf_toupper(const unsigned char *str, int len, char *buf) OksiD
\end{verbatim}

\texttt{fl_tolower()} and \texttt{fl_toupper()} convert a single Unicode character from upper to lower case, and vice versa. \texttt{fl_utf_tolower()} and \texttt{fl_utf_toupper()} convert a string of bytes, some of which may be multi-byte UTF-8 encodings of Unicode characters, from upper to lower case, and vice versa.

Warning: to be safe, \texttt{buf} length must be at least \texttt{3*len} [for 16-bit Unicode]

\begin{verbatim}
int fl_utf_strcasecmp(const char *s1, const char *s2) OksiD
int fl_utf_strncasecmp(const char *s1, const char *s2, int n) OksiD
\end{verbatim}

\texttt{fl_utf_strcasecmp()} is a UTF-8 aware string comparison function that converts the strings to lower case Unicode as part of the comparison. \texttt{fl_utf_strncasecmp()} only compares the first \texttt{n} characters [bytes?]

1.15.5 FLTK Unicode Versions of System Calls

\begin{itemize}
  \item int \texttt{fl_access(const char* f, int mode)} OksiD
  \item int \texttt{fl_chmod(const char* f, int mode)} OksiD
  \item int \texttt{fl_execvp(const char* file, char* const* argv)} OksiD
  \item FILE\* \texttt{fl_fopen(const char* f, const char* mode)} OksiD
  \item char\* \texttt{fl_getcwd(char* buf, int maxlen)} OksiD
  \item char\* \texttt{fl_getenv(const char* name)} OksiD
  \item char \texttt{fl_make_path(const char* path)} - returns char? OksiD
  \item void \texttt{fl_make_path_for_file(const char* path)} OksiD
\end{itemize}
• int fl_mkdir(const char * f, int mode) OksiD
• int fl_open(const char * f, int o, ...) OksiD
• int fl_rename(const char * f, const char * t) OksiD
• int fl_rmdir(const char * f) OksiD
• int fl_stat(const char * path, struct stat * buffer) OksiD
• int fl_system(const char * f) OksiD
• int fl_unlink(const char * f) OksiD

TODO:

• more doc on unicode, add links
• write something about filename encoding on OS X...
• explain the fl_utf8… commands
• explain issues with Fl_Preferences

1.16 Constants and Enumerations

Note
This file is not actively maintained any more, but is left here as a reference, until the doxygen documentation is completed.

See also

FL/Enumerations.H.

This appendix lists the enumerations provided in the `<FL/Enumerations.H>` header file, organized by section. Constants whose value are zero are marked with "(0)", this is often useful to know when programming.

1.16.1 Version Numbers

The FLTK version number is stored in a number of compile-time constants:

• FL_MAJOR_VERSION - The major release number, currently 1
• FL_MINOR_VERSION - The minor release number, currently 4
• FL_PATCH_VERSION - The patch release number, currently 0
• FL_VERSION - [Deprecated] A combined floating-point version number for the major, minor, and patch re-lease numbers, currently 1.0400
• FL_API_VERSION - A combined integer version number for the major, minor, and patch release numbers, currently 10400 (use this instead of FL_VERSION, if possible)
• FL_ABI_VERSION - A combined integer version number for the application binary interface (ABI) major, minor, and patch release numbers, currently 10400 (default)

Note
The ABI version (FL_ABI_VERSION) is usually constant throughout one major/minor release version, for in-stance 10300 if FL_API_VERSION is 10304. Hence the ABI is constant if only the patch version is changed. You can change this with configure or CMake though if you want the latest enhancements (called “ABI fea-tures”, see CHANGES).
1.16.2 Events

Events are identified by an FL_Event enumeration value. The following events are currently defined:

- **FL_NO_EVENT** - No event (or an event fltk does not understand) occurred (0).
- **FL_PUSH** - A mouse button was pushed.
- **FL_RELEASE** - A mouse button was released.
- **FL_ENTER** - The mouse pointer entered a widget.
- **FL_LEAVE** - The mouse pointer left a widget.
- **FL_DRAG** - The mouse pointer was moved with a button pressed.
- **FL_FOCUS** - A widget should receive keyboard focus.
- **FL_UNFOCUS** - A widget loses keyboard focus.
- **FL_KEYBOARD** - A key was pressed.
- **FL_CLOSE** - A window was closed.
- **FL_MOVE** - The mouse pointer was moved with no buttons pressed.
- **FL_SHORTCUT** - The user pressed a shortcut key.
- **FL_DEACTIVATE** - The widget has been deactivated.
- **FL_ACTIVATE** - The widget has been activated.
- **FL_HIDE** - The widget has been hidden.
- **FL_SHOW** - The widget has been shown.
- **FL_PASTE** - The widget should paste the contents of the clipboard.
- **FL_SELECTIONCLEAR** - The widget should clear any selections made for the clipboard.
- **FL_MOUSEWHEEL** - The horizontal or vertical mousewheel was turned.
- **FL_DND_ENTER** - The mouse pointer entered a widget dragging data.
- **FL_DND_DRAG** - The mouse pointer was moved dragging data.
- **FL_DND_LEAVE** - The mouse pointer left a widget still dragging data.
- **FL_DND_RELEASE** - Dragged data is about to be dropped.
- **FL_SCREEN_CONFIGURATION_CHANGED** - The screen configuration (number, positions) was changed.
- **FL_FULLSCREEN** - The fullscreen state of the window has changed.

1.16.3 Callback "When" Conditions

The following constants determine when a callback is performed:

- **FL_WHEN_NEVER** - Never call the callback (0).
- **FL_WHEN_CHANGED** - Do the callback only when the widget value changes.
- **FL_WHEN_NOT_CHANGED** - Do the callback whenever the user interacts with the widget.
- **FL_WHEN_RELEASE** - Do the callback when the button or key is released and the value changes.
- **FL_WHEN_ENTER_KEY** - Do the callback when the user presses the ENTER key and the value changes.
- **FL_WHEN_RELEASE_ALWAYS** - Do the callback when the button or key is released, even if the value doesn't change.
- **FL_WHEN_ENTER_KEY_ALWAYS** - Do the callback when the user presses the ENTER key, even if the value doesn't change.
1.16.4 Fl::event_button() Values

The following constants define the button numbers for FL_PUSH and FL_RELEASE events:

- FL_LEFT_MOUSE - the left mouse button
- FL_MIDDLE_MOUSE - the middle mouse button
- FL_RIGHT_MOUSE - the right mouse button

1.16.5 Fl::event_key() Values

The following constants define the non-ASCII keys on the keyboard for FL_KEYBOARD and FL_SHORTCUT events:

- FL_Button - A mouse button; use \texttt{Fl\_Button} + n for mouse button n.
- FL_BackSpace - The backspace key.
- FL_Tab - The tab key.
- FL_Enter - The enter key.
- FL_Pause - The pause key.
- FL_Scroll\_Lock - The scroll lock key.
- FL_Escape - The escape key.
- FL_Home - The home key.
- FL_Left - The left arrow key.
- FL_Up - The up arrow key.
- FL_Right - The right arrow key.
- FL_Down - The down arrow key.
- FL_Page\_Up - The page-up key.
- FL_Page\_Down - The page-down key.
- FL_End - The end key.
- FL_Print - The print (or print-screen) key.
- FL_Insert - The insert key.
- FL_Menu - The menu key.
- FL_Num\_Lock - The num lock key.
- FL_KP - One of the keypad numbers or keys; use \texttt{Fl\_KP} + 'n' for number n and, say, \texttt{Fl\_KP} + '*'.
- FL_KP\_Enter - The enter key on the keypad.
- FL_F - One of the function keys; use \texttt{Fl\_F} + n for function key n.
- FL_Shift\_L - The lefthand shift key.
- FL_Shift\_R - The righthand shift key.
- FL_Control\_L - The lefthand control key.
• FL_Control_R - The righthand control key.
• FL_Caps_Lock - The caps lock key.
• FL_Meta_L - The left meta/Windows key.
• FL_Meta_R - The right meta/Windows key.
• FL_Alt_L - The left alt key.
• FL_Alt_R - The right alt key.
• FL_Delete - The delete key.

### 1.16.6 Fl::event_state() Values

The following constants define bits in the Fl::event_state() value:

• FL_SHIFT - One of the shift keys is down.
• FL_CAPS_LOCK - The caps lock is on.
• FL_CTRL - One of the ctrl keys is down.
• FL_ALT - One of the alt keys is down.
• FL_NUM_LOCK - The num lock is on.
• FL_META - One of the meta/Windows keys is down.
• FL_COMMAND - An alias for FL_CTRL on Windows, X11 and Wayland, or FL_META on MacOS X.
• FL_CONTROL - An alias for FL_META on Windows, X11 and Wayland, or FL_CTRL on MacOS X.
• FL_SCROLL_LOCK - The scroll lock is on.
• FL_BUTTON1 - Mouse button 1 is pushed.
• FL_BUTTON2 - Mouse button 2 is pushed.
• FL_BUTTON3 - Mouse button 3 is pushed.
• FL BUTTONS - Any mouse button is pushed.
• FL_BUTTON(n) - Mouse button n (where n > 0) is pushed.

### 1.16.7 Alignment Values

The following constants define bits that can be used with Fl_Widget::align() to control the positioning of the label:

• FL_ALIGN_CENTER - The label is centered (0).
• FL_ALIGN_TOP - The label is top-aligned.
• FL_ALIGN_BOTTOM - The label is bottom-aligned.
• FL_ALIGN_LEFT - The label is left-aligned.
• FL_ALIGN_RIGHT - The label is right-aligned.
• FL_ALIGN_CLIP - The label is clipped to the widget.
• FL_ALIGN_WRAP - The label text is wrapped as needed.

• FL_ALIGN_TOP_LEFT - The label appears at the top of the widget, aligned to the left.

• FL_ALIGN_TOP_RIGHT - The label appears at the top of the widget, aligned to the right.

• FL_ALIGN_BOTTOM_LEFT - The label appears at the bottom of the widget, aligned to the left.

• FL_ALIGN_BOTTOM_RIGHT - The label appears at the bottom of the widget, aligned to the right.

• FL_ALIGN_LEFT_TOP - The label appears to the left of the widget, aligned at the top. Outside labels only.

• FL_ALIGN_RIGHT_TOP - The label appears to the right of the widget, aligned at the top. Outside labels only.

• FL_ALIGN_LEFT_BOTTOM - The label appears to the left of the widget, aligned at the bottom. Outside labels only.

• FL_ALIGN_RIGHT_BOTTOM - The label appears to the right of the widget, aligned at the bottom. Outside labels only.

• FL_ALIGN_INSIDE - 'or' this with other values to put label inside the widget.

• FL_ALIGN_TEXT_OVER_IMAGE - Label text will appear above the image.

• FL_ALIGN_IMAGE_OVER_TEXT - Label text will be below the image.

• FL_ALIGN_IMAGE_NEXT_TO_TEXT - The image will appear to the left of the text.

• FL_ALIGN_TEXT_NEXT_TO_IMAGE - The image will appear to the right of the text.

• FL_ALIGN_IMAGE_BACKDROP - The image will be used as a background for the widget.

1.16.8 Fonts

The following constants define the standard FLTK fonts:

• FL_HELVETICA - Helvetica (or Arial) normal (0).

• FL_HELVETICA_BOLD - Helvetica (or Arial) bold.

• FL_HELVETICA_ITALIC - Helvetica (or Arial) oblique.

• FL_HELVETICA_BOLD_ITALIC - Helvetica (or Arial) bold-oblique.

• FL_COURIER - Courier normal.

• FL_COURIER_BOLD - Courier bold.

• FL_COURIER_ITALIC - Courier italic.

• FL_COURIER_BOLD_ITALIC - Courier bold-italic.

• FL_TIMES - Times roman.

• FL_TIMES_BOLD - Times bold.

• FL_TIMES_ITALIC - Times italic.

• FL_TIMES_BOLD_ITALIC - Times bold-italic.

• FL_SYMBOL - Standard symbol font.

• FL_SCREEN - Default monospaced screen font.

• FL_SCREEN_BOLD - Default monospaced bold screen font.

• FL_ZAPF_DINGBATS - Zapf-dingbats font.
1.16.9 Colors

The Fl_Color enumeration type holds a FLTK color value. Colors are either 8-bit indexes into a virtual colormap or 24-bit RGB color values. Color indices occupy the lower 8 bits of the value, while RGB colors occupy the upper 24 bits, for a byte organization of RGBI.

1.16.9.1 Color Constants

Constants are defined for the user-defined foreground and background colors, as well as specific colors and the start of the grayscale ramp and color cube in the virtual colormap. Inline functions are provided to retrieve specific grayscale, color cube, or RGB color values.

The following color constants can be used to access the user-defined colors:

- FL_BACKGROUND_COLOR - the default background color
- FL_BACKGROUND2_COLOR - the default background color for text, list, and valuator widgets
- FL_FOREGROUND_COLOR - the default foreground color (0) used for labels and text
- FL_INACTIVE_COLOR - the inactive foreground color
- FL_SELECTION_COLOR - the default selection/highlight color

The following color constants can be used to access the colors from the FLTK standard color cube:

- FL_BLACK
- FL_BLUE
- FL_CYAN
- FL_DARK_BLUE
- FL_DARK_CYAN
- FL_DARK_GREEN
- FL_DARK_MAGENTA
- FL_DARK_RED
- FL_DARK_YELLOW
- FL_GREEN
- FL_MAGENTA
- FL_RED
- FL_WHITE
- FL_YELLOW

The following are named values within the standard grayscale:

- FL_GRAY0
- FL_DARK3
The inline methods for getting a grayscale, color cube, or RGB color value are described in the Colors section of the Drawing Things in FLTK chapter.

### 1.16.10 Cursors

The following constants define the mouse cursors that are available in FLTK. The double-headed arrows are bitmaps provided by FLTK on X, the others are provided by system-defined cursors.

- FL_CURSOR_DEFAULT - the default cursor, usually an arrow (0)
- FL_CURSOR_ARROW - an arrow pointer
- FL_CURSOR_CROSS - crosshair
- FL_CURSOR_WAIT - watch or hourglass
- FL_CURSOR_INSERT - I-beam
- FL_CURSOR_HAND - hand (uparrow on Windows)
- FL_CURSOR_HELP - question mark
- FL_CURSOR_MOVE - 4-pointed arrow
- FL_CURSOR_NS - up/down arrow
- FL_CURSOR_WE - left/right arrow
- FL_CURSOR_NWSE - diagonal arrow
- FL_CURSOR_NESW - diagonal arrow
- FL_CURSOR_NONE - invisible

### 1.16.11 FD "When" Conditions

- FL_READ - Call the callback when there is data to be read.
- FL_WRITE - Call the callback when data can be written without blocking.
- FL_EXCEPT - Call the callback if an exception occurs on the file.
1.16.12 Damage Masks

The following damage mask bits are used by the standard FLTK widgets:

- FL_DAMAGE_CHILD - A child needs to be redrawn.
- FL_DAMAGE_EXPOSE - The window was exposed.
- FL_DAMAGE_SCROLL - The Fl_Scroll widget was scrolled.
- FL_DAMAGE_OVERLAY - The overlay planes need to be redrawn.
- FL_DAMAGE_USER1 - First user-defined damage bit.
- FL_DAMAGE_USER2 - Second user-defined damage bit.
- FL_DAMAGE_ALL - Everything needs to be redrawn.

1.17 GLUT Compatibility

This appendix describes the GLUT compatibility header file supplied with FLTK.

FLTK's GLUT compatibility is based on the original GLUT 3.7 and the follow-on FreeGLUT 2.4.0 libraries.

1.17.1 Using the GLUT Compatibility Header File

You should be able to compile existing GLUT source code by including `<FL/glut.H>` instead of `<GL/glut.h>`. This can be done by editing the source, by changing the -I switches to the compiler, or by providing a symbolic link from GL/glut.h to FL/glut.H.

All files calling GLUT procedures must be compiled with C++. You may have to alter them slightly to get them to compile without warnings, and you may have to rename them to get make to use the C++ compiler.

You must link with the FLTK library. Most of FL/glut.H is inline functions. You should take a look at it (and maybe at test/ glpuzzle.cxx in the FLTK source) if you are having trouble porting your GLUT program.

This has been tested with most of the demo programs that come with the GLUT and FreeGLUT distributions.

1.17.2 Known Problems

The following functions and/or arguments to functions are missing, and you will have to replace them or comment them out for your code to compile:

- glutGet (GLUT_ELAPSED_TIME)
- glutGet (GLUT_SCREEN_HEIGHT_MM)
- glutGet (GLUT_SCREEN_WIDTH_MM)
- glutGet (GLUT_WINDOW_NUM_CHILDREN)
- glutInitDisplayMode (GLUT_LUMINANCE)
- glutKeyboardUpFunc (void (*callback)(unsigned char, int x, int y))
• glutLayerGet(GLUT_HAS_OVERLAY)
• glutLayerGet(GLUT_LAYER_IN_USE)
• glutPushWindow()
• glutSetColor(), glutGetColor(), glutCopyColormap()
• glutVideoResize() missing.
• glutWarpPointer()
• glutWindowStatusFunc()
• Spaceball, buttonbox, dials, and tablet functions

Most of the symbols/enumerations have different values than GLUT uses. This will break code that relies on the actual values. The only symbols guaranteed to have the same values are true/false pairs like GLUT_DOWN and GLUT_UP, mouse buttons GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON, and GLUT_KEY_F1 thru GLUT_KEY_F12.

The strings passed as menu labels are not copied.

glutPostRedisplay() does not work if called from inside a display function. You must use glutIdleFunc() if you want your display to update continuously.

glutSwapBuffers() does not work from inside a display function. This is on purpose, because FLTK swaps the buffers for you.

glutUseLayer() does not work well, and should only be used to initialize transformations inside a resize callback. You should redraw overlays by using glutOverlayDisplayFunc().

Overlays are cleared before the overlay display function is called. glutLayerGet(GLUT_OVERLAY_DAMAGE) always returns true for compatibility with some GLUT overlay programs. You must rewrite your code so that gl_color() is used to choose colors in an overlay, or you will get random overlay colors.

glutSetCursor(GLUT_CURSOR_FULL_CROSSHAIR) just results in a small crosshair.

The fonts used by glutBitmapCharacter() and glutBitmapWidth() may be different.

glutInit(argc, argv) will consume different switches than GLUT does. It accepts the switches recognized by Fl::args(), and will accept any abbreviation of these switches (such as "-di" for "-display").

### 1.17.3 Mixing GLUT and FLTK Code

You can make your GLUT window a child of a Fl_Window with the following scheme. The biggest trick is that GLUT insists on a call to show() the window at the point it is created, which means the Fl_Window parent window must already be shown.

- Don't call glutInit().
- Create your Fl_Window, and any FLTK widgets. Leave a blank area in the window for your GLUT window.
- show() the Fl_Window. Perhaps call show(argc, argv).
- Call window->begin() so that the GLUT window will be automatically added to it.
- Use glutInitWindowSize() and glutInitWindowPosition() to set the location in the parent window to put the GLUT window.
- Put your GLUT code next. It probably does not need many changes. Call window->end() immediately after the glutCreateWindow()!
- You can call either glutMainLoop(), Fl::run(), or loop calling Fl::wait() to run the program.
1.17.4 class Fl_Glut_Window

1.17.4.1 Class Hierarchy

Fl_Gl_Window
    +----Fl_Glut_Window

1.17.4.2 Include Files

#include <FL/glut.H>

1.17.4.3 Description

Each GLUT window is an instance of this class. You may find it useful to manipulate instances directly rather than
use GLUT window id's. These may be created without opening the display, and thus can fit better into FLTK's
method of creating windows.

The current GLUT window is available in the global variable glut_window.

new Fl_Glut_Window(...) is the same as glutCreateWindow() except it does not show() the
window or make the window current.

window->make_current() is the same as glutSetWindow(number). If the window has not had
show() called on it yet, some functions that assume an OpenGL context will not work. If you do show() the
window, call make_current() again to set the context.

∼Fl_Glut_Window() is the same as glutDestroyWindow().

1.17.4.4 Members

The Fl_Glut_Window class contains several public members that can be altered directly:

<table>
<thead>
<tr>
<th>member</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>display</td>
<td>A pointer to the function to call to draw the normal planes.</td>
</tr>
<tr>
<td>entry</td>
<td>A pointer to the function to call when the mouse moves into or out of the window.</td>
</tr>
<tr>
<td>keyboard</td>
<td>A pointer to the function to call when a regular key is pressed.</td>
</tr>
<tr>
<td>menu[3]</td>
<td>The menu to post when one of the mouse buttons is pressed.</td>
</tr>
<tr>
<td>mouse</td>
<td>A pointer to the function to call when a button is pressed or released.</td>
</tr>
<tr>
<td>motion</td>
<td>A pointer to the function to call when the mouse is moved with a button down.</td>
</tr>
<tr>
<td>overlaydisplay</td>
<td>A pointer to the function to call to draw the overlay planes.</td>
</tr>
<tr>
<td>passivemotion</td>
<td>A pointer to the function to call when the mouse is moved with no buttons down.</td>
</tr>
<tr>
<td>reshape</td>
<td>A pointer to the function to call when the window is resized.</td>
</tr>
<tr>
<td>special</td>
<td>A pointer to the function to call when a special key is pressed.</td>
</tr>
<tr>
<td>visibility</td>
<td>A pointer to the function to call when the window is iconified or restored (made visible.)</td>
</tr>
</tbody>
</table>
1.17.4.5 Methods

\texttt{Fl\_Glut\_Window::Fl\_Glut\_Window(int x, int y, int w, int h, const char \ast title = 0)}
\texttt{Fl\_Glut\_Window::Fl\_Glut\_Window(int w, int h, const char \ast title = 0)}

The first constructor takes 4 int arguments to create the window with a preset position and size. The second constructor with 2 arguments will create the window with a preset size, but the window manager will choose the position according to its own whims.

\texttt{virtual Fl\_Glut\_Window::\textasciitilde Fl\_Glut\_Window()}

Destroys the GLUT window.

\texttt{void Fl\_Glut\_Window::make\_current()}

Switches all drawing functions to the GLUT window.

1.18 Forms Compatibility

This appendix describes the Forms compatibility included with FLTK.

\begin{center}
\textbf{Warning: The Forms compatibility is deprecated and no longer maintained since FLTK 1.3.0 and is likely to be removed completely in FLTK 1.4 or 1.5}
\end{center}

1.18.1 Importing Forms Layout Files

\texttt{FLUID} can read the \texttt{.fd} files put out by all versions of Forms and XForms fdesign. However, it will mangle them a bit, but it prints a warning message about anything it does not understand. \texttt{FLUID} cannot write fdesign files, so you should save to a new name so you don't write over the old one.

You will need to edit your main code considerably to get it to link with the output from \texttt{FLUID}. If you are not interested in this you may have more immediate luck with the forms compatibility header, \texttt{<FL/forms.H>}.

1.18.2 Using the Compatibility Header File

You should be able to compile existing Forms or XForms source code by changing the include directory switch to your compiler so that the \texttt{forms.h} file supplied with FLTK is included. The \texttt{forms.h} file simply pulls in \texttt{<FL/forms.H>} so you don't need to change your source code. Take a look at \texttt{<FL/forms.H>} to see how it works, but the basic trick is lots of inline functions. Most of the XForms demo programs work without changes.
You will also have to compile your Forms or XForms program using a C++ compiler. The FLTK library does not provide C bindings or header files.

Although FLTK was designed to be compatible with the GL Forms library (version 0.3 or so), XForms has bloated severely and its interface is X-specific. Therefore, XForms compatibility is no longer a goal of FLTK. Compatibility was limited to things that were free, or that would add code that would not be linked in if the feature is unused, or that was not X-specific.

To use any new features of FLTK, you should rewrite your code to not use the inline functions and instead use "pure" FLTK. This will make it a lot cleaner and make it easier to figure out how to call the FLTK functions. Unfortunately this conversion is harder than expected and even Digital Domain's inhouse code still uses forms.h a lot.

1.18.3 Problems You Will Encounter

Many parts of XForms use X-specific structures like XEvent in their interface. I did not emulate these! Unfortunately these features (such as the "canvas" widget) are needed by most large programs. You will need to rewrite these to use FLTK subclasses.

Fl_Free widgets emulate the old Forms "free" widget. It may be useful for porting programs that change the handle() function on widgets, but you will still need to rewrite things.

Fl_Timer widgets are provided to emulate the XForms timer. These work, but are quite inefficient and inaccurate compared to using Fl::add_timeout().

All instance variables are hidden. If you directly refer to the x, y, w, h, label, or other fields of your Forms widgets you will have to add empty parenthesis after each reference. The easiest way to do this is to globally replace "->x" with "->x()", etc. Replace "boxtpe" with "box()".

const char* arguments to most FLTK methods are simply stored, while Forms would strdup() the passed string. This is most noticeable with the label of widgets. Your program must always pass static data such as a string constant or malloc'd buffer to label(). If you are using labels to display program output you may want to try the Fl_Output widget.

The default fonts and sizes are matched to the older GL version of Forms, so all labels will draw somewhat larger than an XForms program does.

fdesign outputs a setting of a "fdui" instance variable to the main window. I did not emulate this because I wanted all instance variables to be hidden. You can store the same information in the user_data() field of a window. To do this, search through the fdesign output for all occurrences of "->fdui" and edit to use "->user_data()" instead. This will require casts and is not trivial.

The prototype for the functions passed to fl_add_timeout() and fl_set_idle_callback() callback are different.

All the following XForms calls are missing:

- FL_REVISION, fl_library_version()
- FL_RETURN_DBLCLICK (use Fl::event_clicks())
- fl_add_signal_callback()
- fl_set_form_atactivate() fl_set_form_atdeactivate()
- fl_set_form_property()
- fl_set_app_mainform(), fl_get_app_mainform()
- `fl_set_form_minsize()`, `fl_set_form_maxsize()`
- `fl_set_form_event_cmask()`, `fl_get_form_event_cmask()`
- `fl_set_form_dblbuffer()`, `fl_set_object_dblbuffer()` (use an `Fl_Double_Window` instead)
- `fl_adjust_form_size()`
- `fl_register_raw_callback()`
- `fl_set_object_bw()`, `fl_set_border_width()`
- `fl_set_object_resize()`, `fl_set_object_gravity()`
- `fl_set_object_shortcutkey()`
- `fl_set_object_automatic()`
- `fl_get_object_bbox()` (maybe FLTK should do this)
- `fl_set_object_prehandler()`, `fl_set_object_posthandler()`
- `fl_enumerate_fonts()`
- **Most drawing functions**
  - `fl_set_coordunit()` (FLTK uses pixels all the time)
  - `fl_ringbell()`
  - `fl_gettime()`
  - `fl_win*()` (all these functions)
  - `fl_initialize(argc,argv,x,y,z)` ignores last 3 arguments
  - `fl_read_bitmapfile()`, `fl_read_pixmapfile()`
  - `fl_addto_browser_chars()`
  - `FL_MENU_BUTTON` just draws normally
  - `fl_set_bitmapbutton_file()`, `fl_set_pixmapbutton_file()`
- **FL_CANVAS objects**
  - `FL_DIGITAL_CLOCK` (comes out analog)
  - `fl_create_bitmap_cursor()`, `fl_set_cursor_color()`
  - `fl_set_dial_angles()`
  - `fl_show_oneliner()`
  - `fl_set_choice_shortcut(a,b,c)`
- **command log**
  - Only some of file selector is emulated
  - `FL_DATE_INPUT`
  - `fl_pup*()` (all these functions)
- **textbox object** (should be easy but I had no sample programs)
- **xyplot object**
1.18 Forms Compatibility

1.18.4 Additional Notes

These notes were written for porting programs written with the older IRISGL version of Forms. Most of these problems are the same ones encountered when going from old Forms to XForms:

Does Not Run In Background

The IRISGL library always forked when you created the first window, unless "foreground()" was called. FLTK acts like "foreground()" is called all the time. If you really want the fork behavior do "if (fork()) exit(0)" right at the start of your program.

You Cannot Use IRISGL Windows or fl_queue

If a Forms (not XForms) program if you wanted your own window for displaying things you would create a IRISGL window and draw in it, periodically calling Forms to check if the user hit buttons on the panels. If the user did things to the IRISGL window, you would find this out by having the value FL_EVENT returned from the call to Forms.

None of this works with FLTK. Nor will it compile, the necessary calls are not in the interface.

You have to make a subclass of Fl_Gl_Window and write a draw() method and handle() method. This may require anywhere from a trivial to a major rewrite.

If you draw into the overlay planes you will have to also write a draw_overlay() method and call redraw_overlay() on the OpenGL window.

One easy way to hack your program so it works is to make the draw() and handle() methods on your window set some static variables, storing what event happened. Then in the main loop of your program, call Fl::wait() and then check these variables, acting on them as though they are events read from fl_queue.

You Must Use OpenGL to Draw Everything

The file <FL/gl.h> defines replacements for a lot of IRISGL calls, translating them to OpenGL. There are much better translators available that you might want to investigate.

You Cannot Make Forms Subclasses

Programs that call fl_make_object or directly setting the handle routine will not compile. You have to rewrite them to use a subclass of Fl_Widget. It is important to note that the handle() method is not exactly the same as the handle() function of Forms. Where a Forms handle() returned non-zero, your handle() must call do_callback(). And your handle() must return non-zero if it "understood" the event.

An attempt has been made to emulate the "free" widget. This appears to work quite well. It may be quicker to modify your subclass into a "free" widget, since the "handle" functions match.

If your subclass draws into the overlay you are in trouble and will have to rewrite things a lot.

You Cannot Use <device.h>

If you have written your own "free" widgets you will probably get a lot of errors about "getvaluator". You should substitute:
Anything else in getvaluator and you are on your own...

Font Numbers Are Different

The "style" numbers have been changed because I wanted to insert bold-italic versions of the normal fonts. If you use Times, Courier, or Bookman to display any text you will get a different font out of FLTK. If you are really desperate to fix this use the following code:

```c
fl_font_name(3,"*courier-medium-r-no*"),
fl_font_name(4,"*courier-bold-r-no*"),
fl_font_name(5,"*courier-medium-o-no*"),
fl_font_name(6,"*times-medium-r-no*"),
fl_font_name(7,"*times-bold-r-no*"),
fl_font_name(8,"*times-medium-i-no*"),
fl_font_name(9,"*bookman-light-r-no*"),
fl_font_name(10,"*bookman-demi-r-no*"),
fl_font_name(11,"*bookman-light-i-no*"),
```

1.19 Operating System Issues

This appendix describes the operating system specific interfaces in FLTK:

- Accessing the OS Interfaces
- The Wayland/X11 hybrid library
- The UNIX (X11) Interface
- The Windows Interface
- The Apple OS X Interface
- The Wayland Interface

1.19.1 Accessing the OS Interfaces

All programs that need to access the operating system specific interfaces must include the following header file:

```c
#include <FL/platform.H>
```

This header file will define the appropriate interface for your environment. The pages that follow describe the functionality that is provided for each operating system.

Note

These definitions used to be in FL/x.H up to FLTK 1.3.x. Usage of FL/x.H is deprecated since FLTK 1.4.0. You should replace all references of FL/x.H with FL/platform.H if your target is FLTK 1.4 or later. FL/x.H will be retained for backwards compatibility for some releases but will be removed in a later (not yet specified) FLTK release.
1.19 Operating System Issues

WARNING:
The interfaces provided by this header file may change radically in new FLTK releases. Use them only when an existing generic FLTK interface is not sufficient.

1.19.2 The Wayland/X11 hybrid library

By default, the FLTK library is, under Linux and Unix, a Wayland/X11 hybrid which can run FLTK-based apps as Wayland clients or as X11 clients. The choice between running an app as a Wayland or an X11 client is done as follows, when the app runs function `fl_open_display()` (that function can be called explicitly by the app or implicitly by FLTK, for example the first time an `Fl_Window` is show()'):

- if the app contains a global boolean variable named `fl_disable_wayland` and this variable is true, X11 is used;
- if environment variable `FLTK_BACKEND` is not defined, Wayland is used if a Wayland compositor is available, otherwise X11 is used;
- if `$FLTK_BACKEND` equals "wayland", the library makes the app a Wayland client, and stops with error if no Wayland compositor is available;
- if `$FLTK_BACKEND` equals "x11", the library makes the app an X11 client even if a Wayland compositor is available.

The first condition listed above is meant to facilitate conversion of code written for FLTK 1.3.x and containing X11-specific code; add this single statement anywhere in the app's source code:

```c
FL_EXPORT bool fl_disable_wayland = true;
```

and the app will always run as an X11 client.

After function `fl_open_display()` has been called, exactly one of the functions `fl_wl_display()` and `fl_x11_display()` returns a non-NULL value. When the former function does, the app runs as a Wayland client, and Wayland-specific functions and symbols described below (The Wayland Interface) can be used, whereas X11-specific functions and symbols cannot. Otherwise, the app runs as an X11 client, and only X11-specific functions and symbols below (The UNIX (X11) Interface) can be used.

Because a single app can be expected to run either Wayland or X11, it's necessary to use distinct names for global variables and functions in the X11- and the Wayland-specific source code.

Non-default configurations of the FLTK library under Linux/Unix are described in file README.Wayland.txt.

1.19.3 The UNIX (X11) Interface

Cross-platform applications should bracket X11-specific source code between `#if defined(FLTK_USE_X11) / #endif` and should ensure function `fl_x11_display()` returns non-NULL before calling X11-specific functions and using X11-specific symbols.

The UNIX interface provides access to the X Window System state information and data structures.
1.19.3.1 Handling Other X Events

void Fl::add_handler(int (∗f)(int))

Installs a function to parse unrecognized events. If FLTK cannot figure out what to do with an event, it calls each of these functions (most recent first) until one of them returns non-zero. If none of them returns non-zero then the event is ignored.

FLTK calls this for any X events it does not recognize, or X events with a window ID that FLTK does not recognize. You can look at the X event in the fl_xevent variable.

The argument is the FLTK event type that was not handled, or zero for unrecognized X events. These handlers are also called for global shortcuts and some other events that the widget they were passed to did not handle, for example FL_SHORTCUT.

extern XEvent ∗fl_xevent

This variable contains the most recent X event.

extern ulong fl_event_time

This variable contains the time stamp from the most recent X event that reported it; not all events do. Many X calls like cut and paste need this value.

Window fl_xid(const Fl_Window ∗)

Returns the XID for a window, or zero if not shown().

Deprecated Kept for compatibility with FLTK versions before 1.4. Use preferentially fl_x11_xid(const Fl_Window ∗) with versions 1.4 and above.

Fl_Window ∗fl_find(ulong xid)

Returns the Fl_Window that corresponds to the given XID, or NULL if not found. This function uses a cache so it is slightly faster than iterating through the windows yourself.

Deprecated Kept for compatibility with FLTK versions before 1.4. Use preferentially fl_x11_find(Window) with versions 1.4 and above.

int fl_handle(const XEvent &)

This call allows you to supply the X events to FLTK, which may allow FLTK to cooperate with another toolkit or library. The return value is non-zero if FLTK understood the event. If the window does not belong to FLTK and the add_handler() functions all return 0, this function will return false.

Besides feeding events your code should call Fl::flush() periodically so that FLTK redraws its windows.

This function will call the callback functions. It will not return until they complete. In particular, if a callback pops up a modal window by calling fl_ask(), for instance, it will not return until the modal function returns.
1.19.3.2 Drawing using Xlib

The following global variables are set before Fl_Widget::draw() is called, or by Fl_Window::make_current():

```c
extern Display *fl_display; // for compatibility with previous FLTK versions
extern Display *fl_x11_display(); // preferred access starting with FLTK 1.4
extern Window fl_window;
extern GC fl_gc; // for compatibility with previous FLTK versions
extern GC fl_x11_gc(); // preferred access starting with FLTK 1.4
extern int fl_screen;
extern XVisualInfo *fl_visual;
extern Colormap fl_colormap;
```

You must use them to produce Xlib calls. Don’t attempt to change them. A typical X drawing call is written like this:

```c
XDrawSomething(fl_display, fl_window, fl_gc, ...);
```

Other information such as the position or size of the X window can be found by looking at Fl_Window::current(), which returns a pointer to the Fl_Window being drawn.

```c
unsigned long fl_xpixel(Fl_Color i)
unsigned long fl_xpixel(uchar r, uchar g, uchar b)
```

Returns the X pixel number used to draw the given FLTK color index or RGB color. This is the X pixel that fl_color() would use.

```c
int fl_parse_color(const char *p, uchar &r, uchar &g, uchar &b)
```

Convert a name into the red, green, and blue values of a color by parsing the X11 color names. On other systems, fl_parse_color() can only convert names in hexadecimal encoding, for example #ff8083.

```c
extern XFontStruct *fl_xftfont
```

Points to the font selected by the most recent fl_font(). This is not necessarily the current font of fl_gc, which is not set until fl_draw() is called. If FLTK was compiled with Xt support, fl_xftfont will usually be 0 and fl_xftfont will contain a pointer to the XftFont structure instead.

```c
extern void *fl_xftfont
```

If FLTK was compiled with Xt support enabled, fl_xftfont points to the xft font selected by the most recent fl_font(). Otherwise it will be 0. fl_xftfont should be cast to XftFont*.

1.19.3.3 Changing the Display, Screen, or X Visual

FLTK uses only a single display, screen, X visual, and X colormap. This greatly simplifies its internal structure and makes it much smaller and faster. You can change which it uses by setting global variables before the first Fl_Window::show() is called. You may also want to call Fl::visual(), which is a portable interface to get a full color and/or double buffered visual.

```c
int Fl::display(const char *)
```
Set which X display to use. This actually does `putenv("DISPLAY=...")` so that child programs will display on the same screen if called with `exec()`. This must be done before the display is opened. This call is provided under MacOS and Windows but it has no effect.

```c
extern Display *fl_display
```

The open X display. This is needed as an argument to most Xlib calls. Don't attempt to change it! This is `NULL` before the display is opened.

```c
void fl_open_display()
```

Opens the display. Does nothing if it is already open. This will make sure `fl_display` is non-zero. You should call this if you wish to do X calls and there is a chance that your code will be called before the first `show()` of a window.

This may call `Fl::abort()` if there is an error opening the display.

```c
void fl_x11_use_display(Display *d)
```

Directs FLTK to use a pre-established X11 connection.

```c
void fl_close_display()
```

This closes the X connection. You do not need to call this to exit, and in fact it is faster to not do so! It may be useful to call this if you want your program to continue without the X connection. You cannot open the display again, and probably cannot call any FLTK functions.

```c
extern int fl_screen
```

Which screen number to use. This is set by `fl_open_display()` to the default screen. You can change it by setting this to a different value immediately afterwards. It can also be set by changing the last number in the `Fl::display()` string to "host:0.#".

```c
extern XVisualInfo *fl_visual
```
```c
extern Colormap fl_colormap
```

The visual and colormap that FLTK will use for all windows. These are set by `fl_open_display()` to the default visual and colormap. You can change them before calling `show()` on the first window. Typical code for changing the default visual is:

```c
Fl::args(argc, argv); // do this first so $DISPLAY is set
fl_open_display();
fl_visual = find_a_good_visual(fl_display, fl_screen);
if (!fl_visual) Fl::abort("No good visual");
fl_colormap = make_a_colormap(fl_display, fl_visual->visual, fl_visual->depth);
// it is now ok to show() windows:
window->show(argc, argv);
```
1.19 Operating System Issues

1.19.3.4 Using a Subclass of Fl_Window for Special X Stuff

FLTK can manage an X window on a different screen, visual and/or colormap, you just can’t use FLTK’s drawing routines to draw into it. But you can write your own draw() method that uses Xlib (and/or OpenGL) calls only.

FLTK can also manage XID’s provided by other libraries or programs, and call those libraries when the window needs to be redrawn.

To do this, you need to make a subclass of Fl_Window and override some of these virtual functions:

virtual void Fl_Window::show()

If the window is already shown() this must cause it to be raised, this can usually be done by calling Fl_Window::show(). If not shown() your implementation must call either Fl_X::set_xid() or Fl_X::make_xid().

An example:

```c
void MyWindow::show() {
  if (shown()) {Fl_Window::show(); return;} // you must do this!
  fl_open_display(); // necessary if this is first window
  // we only calculate the necessary visual colormap once:
  static XVisualInfo *visual;
  static colormap;
  if (!visual) {
    visual = figure_out_visual();
    colormap = XCreateColormap(fl_display, RootWindow(fl_display,fl_screen),
      vis->visual, AllocNone);
  }
  Fl_X::make_xid(this, visual, colormap);
}
```

Fl_X *Fl_X::set_xid(Fl_Window*, Window xid)

Allocate a hidden class called an Fl_X, put the XID into it, and set a pointer to it from the Fl_Window. This causes Fl_Window::shown() to return true.

void Fl_X::make_xid(Fl_Window*, XVisualInfo* = fl_visual, Colormap = fl_colormap)

This static method does the most onerous parts of creating an X window, including setting the label, resize limitations, etc. It then does Fl_X::set_xid() with this new window and maps the window.

virtual void Fl_Window::flush()

This virtual function is called by Fl::flush() to update the window. For FLTK’s own windows it does this by setting the global variables fl_window and fl_gc and then calling the draw() method. For your own windows you might just want to put all the drawing code in here.
The X region that is a combination of all damage() calls done so far is in Fl_X::flx(this)->region. If NULL then you should redraw the entire window. The undocumented function fl_clip_region(←XRegion) will initialize the FLTK clip stack with a region or NULL for no clipping. You must set region to NULL afterwards as fl_clip_region() will own and delete it when done.

If damage() & FL_DAMAGE_EXPOSE then only X expose events have happened. This may be useful if you have an undamaged image (such as a backing buffer) around.

Here is a sample where an undamaged image is kept somewhere:
void MyWindow::flush() {  
  fl_clip_region(Fl_X::flx(this)->region);  
  Fl_X::flx(this)->region = 0;  
  if (damage() != 2) {... draw things into backing store ...}  
  ... copy backing store to window ...  
}

Note
For compatibility with FLTK versions before 1.4, member function Fl_X::flx(Fl_Window*) can also be written Fl_X::i(Fl_Window*).

virtual void Fl_Window::hide()

Destroy the window server copy of the window. Usually you will destroy contexts, pixmaps, or other resources used by the window, and then call Fl_Window::hide() to get rid of the main window identified by xid(). If you override this, you must also override the destructor as shown:
void MyWindow::hide() {  
  if (mypixmap) {  
    XFreePixmap(fl_display,mypixmap);  
    mypixmap = 0;  
  }  
  Fl_Window::hide(); // you must call this  
}

virtual void Fl_Window::~Fl_Window()  

Because of the way C++ works, if you override hide() you must override the destructor as well (otherwise only the base class hide() is called):
MyWindow::~MyWindow() {  
  hide();  
}

Note
Access to the Fl_X hidden class requires to #define FL_INTERNALS before compilation.
1.19.3.5 Setting the Icon of a Window

FLTK recommends to set window icons using these platform-independent methods: \texttt{Fl\_Window::icon(const Fl\_RGB\_Image *)} and \texttt{Fl\_Window::icons(const Fl\_RGB\_Image *[], int)}. See also methods setting default window icons \texttt{Fl\_Window::default\_icon(const Fl\_RGB\_Image *)} and \texttt{Fl\_Window::default\_icons(const Fl\_RGB\_Image *[], int)}.

FLTK on X11 also supports, for backward compatibility, use of the deprecated method \texttt{Fl\_Window::icon(const void *)} as follows:

Sets the icon for the window to the passed pointer. You will need to cast the icon \texttt{Pixmap} to a \texttt{char *} when calling this method. To set a monochrome icon using a bitmap compiled with your application use:

```c
#include "icon.xbm"

fl_open_display(); // needed if display has not been previously opened

Pixmap p = XCreateBitmapFromData(fl_display, DefaultRootWindow(fl_display),
   icon_bits, icon_width, icon_height);

window->icon((const void*)p);
```

To use a multi-colored icon, the XPM format and library should be used as follows:

```c
#include <X11/xpm.h>
#include "icon.xpm"

fl_open_display(); // needed if display has not been previously opened

Pixmap p, mask;

XpmCreatePixmapFromData(fl_display, DefaultRootWindow(fl_display),
   icon_xpm, &p, &mask, NULL);

window->icon((const void *)p);
```

When using the Xpm library, be sure to include it in the list of libraries that are used to link the application (usually ".\texttt{Xpm}"").

\begin{note}
You must call \texttt{Fl\_Window::show(int argc, char** argv)} for the icon to be used. The \texttt{Fl\_Window::show()} method does not bind the icon to the window.
\end{note}

Any window icon must be set with the above methods before the window is shown.

1.19.3.6 X Resources

When the \texttt{Fl\_Window::show(int argc, char** argv)} method is called, FLTK looks for the following X resources:

- \texttt{background} - The default background color for widgets (color).
- \texttt{dndTextOps} - The default setting for drag and drop text operations (boolean).
- \texttt{foreground} - The default foreground (label) color for widgets (color).
- \texttt{scheme} - The default scheme to use (string).
• `selectBackground` - The default selection color for menus, etc. (color).

• `Text.background` - The default background color for text fields (color).

• `tooltips` - The default setting for tooltips (boolean).

• `visibleFocus` - The default setting for visible keyboard focus on non-text widgets (boolean).

Resources associated with the first window's `Fl_Window::xclass()` string are queried first, or if no class has been specified then the class "fltk" is used (e.g. `fltk.background`). If no match is found, a global search is done (e.g. `∗background`).

1.19.3.7 Display Scaling Factor

FLTK uses the value of the Xft.dpi resource divided by 96. to initialize the display scaling factor. That is also what is done by the gnome and KDE desktops.

1.19.4 The Windows Interface

Cross-platform applications should bracket Windows-specific source code between `#ifdef _WIN32` / `#endif`.

The Windows interface provides access to the Windows GDI state information and data structures.

1.19.4.1 Using filenames with non-ASCII characters

In FLTK, all strings, including filenames, are UTF-8 encoded. The utility functions `fl_fopen()` and `fl_open()` allow to open files potentially having non-ASCII names in a cross-platform fashion, whereas the standard fopen()/open() functions fail to do so.

1.19.4.2 Responding to WM_QUIT

FLTK will intercept WM_QUIT messages that are directed towards the thread that runs the main loop. These are converted to SIGTERM signals via `raise()`. This allows you to deal with outside termination requests with the same code on both Windows and UNIX systems. Other processes can send this message via `PostThreadMessage()` in order to request, rather than force your application to terminate.
1.19 Operating System Issues

1.19.4.3 Handling Other Windows API Messages

By default a single WNDCLASSX called “FLTK” is created. All Fl_Window’s are of this class unless you use Fl_Window::xclass(). The window class is created the first time Fl_Window::show() is called.

You can probably combine FLTK with other libraries that make their own window classes. The easiest way is to call Fl::wait(), as it will call DispatchMessage() for all messages to the other windows. If necessary you can let the other library take over as long as it calls DispatchMessage(), but you will have to arrange for the function Fl::flush() to be called regularly so that widgets are updated, timeouts are handled, and the idle functions are called.

extern MSG fl_msg

This variable contains the most recent message read by GetMessage(), which is called by Fl::wait(). This may not be the most recent message sent to an FLTK window, because silly Windows calls the handle procedures directly for some events (sigh).

void Fl::add_handler(int (*f)(int))

Installs a function to parse unrecognized messages sent to FLTK windows. If FLTK cannot figure out what to do with a message, it calls each of these functions (most recent first) until one of them returns non-zero. The argument passed to the functions is the FLTK event that was not handled or zero for unknown messages. If all the handlers return zero then FLTK calls DefWindowProc().

HWND fl_xid(const Fl_Window *)

Returns the window handle for a Fl_Window, or zero if not shown().

Fl_Window *fl_find(HWND xid)

Returns the Fl_Window that corresponds to the given window handle, or NULL if not found. This function uses a cache so it is slightly faster than iterating through the windows yourself.

1.19.4.4 Drawing Things Using the Windows GDI

When the virtual function Fl_Widget::draw() is called, FLTK stores all the extra arguments you need to make a proper GDI call in some global variables:

extern HINSTANCE fl_display; // for compatibility with previous FLTK versions
extern HINSTANCE fl_win32_display(); // preferred access starting with FLTK 1.4
extern HWND fl_window;
extern HDC fl_gc; // for compatibility with previous FLTK versions
extern HDC fl_win32_gc(); // preferred access starting with FLTK 1.4
COLORREF fl_RGB();
HPEN fl_pen();
HBRUSH fl_brush();

These global variables are set before Fl_Widget::draw() is called, or by Fl_Window::make_current(). You can refer to them when needed to produce GDI calls, but don’t attempt to change them. The functions return GDI objects for the current color set by fl_color() and are created as needed and cached. A typical GDI drawing call is written like this:

DrawSomething(fl_gc, ..., fl_brush());

It may also be useful to refer to Fl_Window::current() to get the window’s size or position.
1.19.4.5 HighDPI support

FLTK apps for the Windows platform are by default "Per-monitor DPI-aware V2". This means that any window automatically adjusts its physical size in relation to the scaling factor of the display where it maps. This also means that all drawings (e.g., text, lines, images) take advantage of the full resolution of the display in use. FLTK apps may also use the manifest mechanism to declare their level of DPI awareness. The FLTK library adapts to the DPI awareness level set in the app's manifest, which can be lower than the default level if the manifest sets it so.

1.19.4.6 Display Scaling Factor

FLTK uses the value given by function GetDpiForMonitor() divided by 96. to initialize the scaling factor of each display in the system. This matches the value of "Change the size of text, apps and other items" found in section “System” subsection “Display” of Windows settings.

1.19.4.7 Setting the Icon of a Window

FLTK recommends to set window icons using these platform-independent methods: Fl_Window::icon(const Fl_RGB_Image *) and Fl_Window::icons(const Fl_RGB_Image *[ ], int). See also methods setting default window icons Fl_Window::default_icon(const Fl_RGB_Image *) and Fl_Window::default_icons(const Fl_RGB_Image *[ ], int).

FLTK on Windows also supports, for backward compatibility, use of the deprecated method Fl_Window::icon(const void *) as follows:
Set the icon for the window to the passed pointer. You will need to cast the HICON handle to a char* when calling this method. To set the icon using an icon resource compiled with your application use:

```
window->icon((const void *)LoadIcon(fl_display, MAKEINTRESOURCE(IDI_ICON)));
```

You can also use the LoadImage() and related functions to load specific resolutions or create the icon from bitmap data.

NOTE:
You must call Fl_Window::show(int argc, char **argv) for the icon to be used. The Fl_Window::show() method does not bind the icon to the window.

Any window icon must be set with the above methods before the window is shown.

1.19.4.8 How to Not Get a MSDOS Console Window

Windows has a really stupid mode switch stored in the executables that controls whether or not to make a console window.

To always get a console window you simply create a console application (the "/SUBSYSTEM:CONSOLE" option for the linker). For a GUI-only application create a Windows application (the="/SUBSYSTEM:WINDOWS" option for the linker).

FLTK includes a WinMain() function that calls the ANSI standard main() entry point for you. This function creates a console window when you use the debug version of the library.

Windows applications without a console cannot write to stdout or stderr, even if they are run from a console window. Any output is silently thrown away. Additionally, Windows applications are run in the background by the console, although you can use "start /wait program" to run them in the foreground.
1.19 Operating System Issues

1.19.4.9 Known Windows Bugs and Problems

The following is a list of known bugs and problems in the Windows version of FLTK:

- If a program is deactivated, `Fl::wait()` does not return until it is activated again, even though many events are delivered to the program. This can cause idle background processes to stop unexpectedly. This also happens while the user is dragging or resizing windows or otherwise holding the mouse down. We were forced to remove most of the efficiency FLTK uses for redrawing in order to get windows to update while being moved. This is a design error in Windows and probably impossible to get around.

- `Fl_Gl_Window::can_do_overlay()` returns true until the first time it attempts to draw an overlay, and then correctly returns whether or not there is overlay hardware.

- `SetCapture` (used by `Fl::grab()`) doesn’t work, and the main window title bar turns gray while menus are popped up.

- Compilation with gcc 3.4.4 and `-Os` exposes an optimisation bug in gcc. The symptom is that when drawing filled circles only the perimeter is drawn. This can for instance be seen in the symbols demo. Other optimisation options such as -O2 and -O3 seem to work OK. More details can be found in STR#1656

1.19.5 The Apple OS X Interface

Cross-platform applications should bracket macOS-specific source code between `#if defined(__APPLE__) && !defined(FLTK_USE_X11) / #endif`. FLTK supports Apple OS X using the Apple Cocoa library. Older versions of MacOS are no longer supported.

Control, Option, and Command Modifier Keys

FLTK maps the Mac ‘control’ key to `FL_CTRL`, the ‘option’ key to `FL_ALT` and the ‘Apple’ key to `FL_META`. Furthermore, `FL_COMMAND` designates the ‘Apple’ key on Mac OS X and the ‘control’ key on other platforms. Keyboard events return the key name in `Fl::event_key()` and the keystroke translation in `Fl::event_text()`. For example, typing Option-Y on a Mac US keyboard will set `FL_ALT` in `Fl::event_state()`, set `Fl::event_key()` to ‘y’ and return the Yen symbol in `Fl::event_text()`.

Right Click simulation with Ctrl Click

The Apple HIG guidelines indicate applications should support ‘Ctrl Click’ to simulate ‘Right Click’ for e.g. context menus, so users with one-button mice and one-click trackpads can still access right-click features. However, paraphrasing Manolo’s comment on the fltk.coredev newsgroup:

- FLTK does /not/ support Ctrl-Click == Right Click itself because Mac OS X event processing doesn’t support this at the system level: the system reports left-clicks with the ctrl modifier when the user ctrl-clicks, and OS X system preferences don’t allow changing this behavior. Therefore, applications must handle simulation of Right Click with Ctrl Click in the application code.
Ian MacArthur provided the following handle() method code snippet showing an example of how to do this:

```c
    case FL_PUSH:
    |
    int btn = Fl::event_button();
    ifndef __APPLE__
    int ev_state = Fl::event_state();
    endif
    /
    // Context menu can be called up in one of two ways: -
    // 1 - right click, as normally used on Windows and Linux
    // 2 - Ctrl + left click, as sometimes used on Mac
    //
    ifndef __APPLE__
    // On apple, check right click, and ctrl+left click
    if ((btn == FL_RIGHT_MOUSE) || (ev_state == (FL_CTRL | FL_BUTTON1)))
    else
    // On other platforms, only check right click as ctrl+left is used for selections
    if (btn == FL_RIGHT_MOUSE)
    endif
    |
    // Did we right click on the object?..
```

There is a thread about this subject on fltk.coredev (Aug 1-14, 2014) entitled “[RFC] Right click emulation for one button mouse on Mac”.

Apple “Quit” Event

When the user presses Cmd-Q or requests a termination of the application, FLTK sends an FL_CLOSE event to all open windows. If any window remains open, the termination request aborts. If all windows close, the application’s event loop terminates, that is, Fl::run() returns. The application can then follow FLTK’s normal termination path executing cleanup code that may be programmed after termination of the event loop, and returning from main(). Function Fl::program_should_quit() allows to detect whether the event loop terminated because of a program termination request.

Apple “Open” Event

Whenever the user drops a file onto an application icon, OS X generates an Apple Event of the type “Open”. You can have FLTK notify you of an Open event by calling the fl_open_callback() function.

```c
    void fl_open_display()
```

Opens the display. Does nothing if it is already open. You should call this if you wish to do Cocoa or Quartz calls and there is a chance that your code will be called before the first show() of a window.

```c
    Window fl_xid(const Fl_Window *)
```

Returns the window reference for an Fl_Window, or NULL if the window has not been shown. This reference is a pointer to an instance of the subclass FLWindow of Cocoa’s NSWindow class.

```c
    Fl_Window *fl_find(Window xid)
```

Generated by Doxygen
Returns the Fl_Window that corresponds to the given window reference, or NULL if not found.

void fl_mac_set_about( Fl_Callback ∗cb, void ∗user_data, int shortcut)

Attaches the callback cb to the "About myprog" item of the system application menu. cb will be called with NULL first argument and user_data second argument. This MacOS-specific function is deprecated in FLTK 1.4 and replaced by Fl_Sys_Menu_Bar::about(Fl_Callback ∗cb, void ∗data) which is cross-platform.

Fl_Sys_Menu_Bar class

The Fl_Sys_Menu_Bar class allows to build menu bars that, on Mac OS X, are placed in the system menu bar (at top-left of display), and, on other platforms, at a user-chosen location of a user-chosen window.

1.19.5.1 Setting the icon of an application

• First, create a .icns file containing several copies of your icon of decreasing sizes. This can be done using the Preview application or the Icon Composer application available in "Graphics Tools for Xcode". To create a high resolution icon file, it is necessary to use the iconutil command-line utility.

• Put your .icns file in the Resources subdirectory of your application bundle.

• Add these two lines to the Info.plist file of your application bundle

  <key>CFBundleIconFile</key>
  <string>foo.icns</string>

replacing foo by your application name. If you use Xcode, just add your .icns file to your application target.

1.19.5.2 Drawing Things Using Quartz

All code inside Fl_Widget::draw() is expected to call Quartz drawing functions. The Quartz coordinate system is flipped to match FLTK's coordinate system. The origin for all drawing is in the top left corner of the enclosing Fl_Window. The function fl_mac_gc() returns the appropriate Quartz 2D drawing environment (of type CGContextRef). For compatibility with previous FLTK versions, deprecated global variable fl_gc gives the same value.

Include FL/platform.H to declare the fl_mac_gc() function (or the fl_gc variable).

1.19.5.3 Internationalization

All FLTK programs contain an application menu with, e.g., the About xxx, Hide xxx, and Quit xxx items. This menu can be internationalized/localized by any of two means.

• using the Fl_Mac_App_Menu class.

• using the standard Mac OS X localization procedure. Create a language-specific .lproj directory (e.g., German.lproj) in the Resources subdirectory of the application bundle. Create therein a Localizable.strings file that translates all menu items to this language. The German Localizable.strings file, for example, contains:

    "About %@" = "Über %@";
    "Print Front Window"="Frontfenster drucken";
    "Services" = "Dienste";
    "Hide %@"="%0 ausblenden";
    "Hide Others"="Andere ausblenden";
    "Show All"="Alle einblenden";
    "Quit %@"="%0 beenden";

Set "Print Front Window" = ""; therein so the application menu doesn't show a "Print Front Window" item. To localize the application name itself, create a file InfoPlist.strings in each .lproj directory and put CFBundleName = "localized name"; in each such file.
1.19.5.4 OpenGL and ’retina’ displays

It is possible to have OpenGL produce graphics at the high pixel resolution allowed by the so-called ‘retina’ displays present on recent Apple hardware. For this, call

```cpp
Fl::use_high_res_GL(1);
```

before any `Fl_Gl_Window` is shown. Also, adapt your `Fl_Gl_Window::draw()` and `Fl_Gl_Window::draw_overlay()` methods replacing

```cpp
glViewport(0, 0, w(), h());
```

by

```cpp
glViewport(0, 0, pixel_w(), pixel_h());
```

making use of the `Fl_Gl_Window::pixel_w()` and `Fl_Gl_Window::pixel_h()` methods that return the width and height of the GL scene in pixels: if the `Fl_Gl_Window` is mapped on a retina display, these methods return twice as much as reported by `Fl_Widget::w()` and `Fl_Widget::h()`; if it’s mapped on a regular display, they return the same values as `w()` and `h()`. These methods dynamically change their values if the window is moved into/out from a retina display. If `Fl::use_high_res_GL(1)` is not called, all `Fl_Gl_Window`'s are drawn at low resolution. These methods are useful on all platforms because `Fl_Gl_Window::w()` and `Fl_Gl_Window::h()` don't return, on HighDPI displays, the quantities in pixels necessary to OpenGL functions.

The `Fl_Gl_Window::pixels_per_unit()` method is useful when the OpenGL code depends on the pixel dimension of the GL scene. This occurs, e.g., if a window's handle() method uses `Fl::event_x()` and `Fl::event_y()` whose returned values should be multiplied by `Fl_Gl_Window::pixels_per_unit()` to obtain the adequate pixel units. This method may also be useful, for example, to adjust the width of a line in a high resolution GL scene.

1.19.5.5 Fl_Double_Window

OS X double-buffers all windows automatically. On OS X, `Fl_Window` and `Fl_Double_Window` are handled internally in the same way.

1.19.5.6 Mac File System Specifics

Resource Forks

FLTK does not access the resource fork of an application. However, a minimal resource fork must be created for OS X applications. Starting with OS X 10.6, resource forks are no longer needed.

Caution (OS X 10.2 and older):

When using UNIX commands to copy or move executables, OS X will NOT copy any resource forks! For copying and moving use `CpMac` and `MvMac` respectively. For creating a tar archive, all executables need to be stripped from their Resource Fork before packing, e.g. "DeRez fluid > fluid.r". After unpacking the Resource Fork needs to be reattached, e.g. "Rez fluid.r -o fluid".
It is advisable to use the Finder for moving and copying and Mac archiving tools like Sit for distribution as they will handle the Resource Fork correctly.

**Mac File Paths**

FLTK uses UTF-8-encoded UNIX-style filenames and paths.

See also

**Mac OS X-specific symbols**

### 1.19.6 The Wayland Interface

Cross-platform applications should bracket Wayland-specific source code between 

```c
#ifdef FLTK_USE_WAYLAND
    ...
#endif
```

and should ensure function `fl_wl_display()` returns non-NULL before calling Wayland-specific functions and using Wayland-specific symbols.

```c
extern struct wl_display *fl_wl_display();
```

After `fl_open_display()` has run, function `fl_wl_display()` returns a pointer to the struct `wl_display` representing the connection between the application and Wayland. For example, `wl_display_get_fd(fl_wl_display())` gives the file descriptor one can use to communicate with the Wayland compositor according to the Wayland protocol.

```c
struct wl_window *fl_wl_xid(const Fl_Window *)
```

Returns a pointer to an FLTK-defined structure holding Wayland-related data created when a window gets show()'n, or NULL if not show()'n.

```c
Fl_Window *fl_wl_find(struct wl_window * wld_win)
```

Returns the Fl_Window that corresponds to the given Window, or NULL if not found.

```c
struct wl_surface *fl_wl_surface(struct wl_window *wld_win)
```

Returns a pointer to the struct `wl_surface` corresponding to a show()'n top-level window or subwindow.

```c
cairo_t *fl_wl_gc()
```

Drawing natively to a Wayland window: Within an overridden `Fl_Widget::draw()` method, or after a call to `Fl_Window::make_current()`, it's possible to draw using the Cairo library. Function `fl_wl_gc()` returns the adequate `cairo_t` value. Regular FLTK coordinates, with top-left origin, are to be used. All FLTK-defined drawing functions (e.g., `fl_rect()`, `fl_draw()`) can be used too.

```c
void fl_close_display()
```

This closes the Wayland connection. You do not need to call this to exit. It may be useful to call this if you want your program to continue without the Wayland connection. You cannot open the display again, and cannot call any FLTK functions.

See also

**wayland.H** for all functions specific of the Wayland platform.
1.19.6.1 HiDPI display support

FLTK Wayland apps automatically scale according to the Wayland-defined, integer-valued scale factor. On a HiDPI display, it's enough to set this factor to 2 for any FLTK app to be drawn using twice as many pixels and thus to be as readable as it is on a regular display. With the gnome and KDE-plasma desktops, that is achieved in the "Settings" section of the "Displays" application, selecting 200 % for the "Scale" parameter. In addition to this, FLTK apps can also be scaled up or down typing ctrl/+/-/0/ and with the FLTK_SCALING_FACTOR environment variable.

1.19.6.2 Window icons

Standard FLTK functions Fl_Window::icon(const Fl_RGB_Image*), Fl_Window::icons(const Fl_RGB_Image*, int), Fl_Window::default_icon(const Fl_RGB_Image*) and Fl_Window::default_icons(const Fl_RGB_Image*, int) have no effect on the Wayland platform. The equivalent of a call to Fl_Window::default_icon(const Fl_RGB_Image*) to set the application-specific window icon can be obtained as follows, using FLTK's editor app as an example:

• create a text file named editor.desktop containing:

```
[Desktop Entry]
Version=1.0
Type=Application
Name=Editor
Name[fr]=Editeur
Comment=FLTK editor
Exec=editor %F
Icon=/path/to/icon/file/editor.svg
MimeType=text/plain
```

• The Name= line therein determines the string displayed when the app runs.

• Optionally, one or more Name[locale]= lines can be used to set locale-specific app names.

• The Icon= line accepts also .png files.

• Put this file in /usr/local/share/applications/ so it's available to all system users or in $ HOME/.local/share/applications/ so it's available to a single user.

1.19.6.3 Window titlebars

Wayland supports both client-side window decoration (CSD), where client applications are responsible for drawing window titlebars, and server-side window decoration (SSD), where the Wayland compositor itself draws window titlebars. Among 4 tested Wayland compositors, Mutter (gnome's compositor) and Weston use CSD mode whereas the KWin and Sway compositors use SSD mode. When running in CSD mode, FLTK uses a library called libdecor to draw titlebars. The libdecor library has been conceived to use various plug-in's to draw titlebars in various fashions intended to match any desktop's preferred titlebar style. FLTK supports drawing titlebars with any libdecor plug-in via an environment variable called LIBDECOR_PLUGIN_DIR which can be given the name of a directory containing the desired plug-in. When LIBDECOR_PLUGIN_DIR is not defined, or points to a directory that doesn't contain a libdecor plug-in, FLTK uses its built-in plug-in to draw titlebars. That is the most common situation, until libdecor plug-in's become available for popular UNIX desktops.

1.20 Migrating Code from FLTK 1.3 to 1.4

This appendix describes the differences between FLTK 1.3.x and FLTK 1.4.x functions and classes and potential requirements to change source code.

We also explain how code can be made compatible so it can be compiled by both FLTK 1.3.x and 1.4.x.

If you need to migrate your code from prior FLTK versions to FLTK 1.4, then you should first consult the relevant appendices in the FLTK 1.3 online documentation or by downloading the FLTK 1.3 documentation. See https://www.fltk.org/doc-1.3/index.html and/or https://www.fltk.org/software.php, respectively.
1.20 Migrating Code from FLTK 1.3 to 1.4

1.20.1 Changes in Header Files

We strive to include only necessary header files in the public headers of the FLTK library to reduce dependencies and hence compile times.

We try to avoid including system header files as far as possible. Known exceptions are `<stdio.h>` where file system structures and functions are visible in the public API, for instance `FILE*`, and sometimes essential header files like `<stdlib.h>` and/or `<stddef.h>`. Some required system headers may be included in platform specific header files like `<FL/platform.H>` or `<FL/platform_types.h>`.

In earlier versions (1.3.x) some of the public FLTK headers included some not strictly required system headers by accident.

The consequence for building user programs with FLTK 1.4 is that if you require a system or FLTK header in your user program that you don't `#include` explicitly but which has been included by FLTK 1.3.x your FLTK 1.3 program may issue compiler errors or warnings about missing header files or missing declarations when compiled with FLTK 1.4.

This is not a fault of FLTK 1.4 but a fault of the source code that did not include all required headers.

In FLTK 1.4 inclusion of `<FL/Fl.H>` is no longer a strict requirement as it was required and documented in FLTK 1.3.x. In FLTK 1.4 you may still need to `#include <FL/Fl.H>` if you are using enumerations or methods of class `Fl` like `Fl::run()` but there are exceptions where this header is included by other FLTK headers, like `Fl_Window.H` and other subclasses.

Suggested solution: include all FLTK and system header files your source code requires explicitly and don't rely on FLTK headers to include a particular header file. If you want your code to be as much as possible compatible with FLTK 1.3.x, then you should `#include <FL/Fl.H>` as required by 1.3.x.

You don't need to include headers of base classes - this is done by all FLTK headers as required. Besides that you need to include some support headers if you use FLTK functions like `fl_choice()` and others. This is described in the function's documentation (if a required header is missing in the docs this is a bug).

If you follow these rules your program will be compatible with both FLTK 1.3.x and FLTK 1.4.x as long as you use only functions and classes defined in FLTK 1.3.

1.20.2 Fl_Preferences

Starting with FLTK 1.3, preference databases are expected to be in UTF-8 encoding. Previous databases were stored in the current character set or code page which renders them incompatible for text entries using international characters.

Starting with FLTK 1.4, searching a valid path to store the preference files has changed slightly. Please see `Fl_Preferences::Fl_Preferences(Root, const char*, const char*)` for details.

On Unix/Linux platforms new FLTK preference files are stored using the `XDG Base Directory Specification` which means in essence that user preference files are stored in the user's home directory under the subdirectory `.config`, i.e. in `$HOME/.config/ftlk.org/` rather than `$HOME/.fltk/ftlk.org/`. Existing preference files are still found and used, hence this new location is optional.

You may want to move the preference files from their old locations to their new locations as documented in `Fl_Preferences::Fl_Preferences(Root, const char*, const char*)`.

New `Fl_Preferences` types `Fl_Preferences::USER_L`, `Fl_Preferences::SYSTEM_L` and some more combinations with "_L" suffix have been defined to make preference files independent of the current locale. This is particularly important for floating point data which is stored in text form with varying decimal separator depending on the locale (either '.' or ','). You may want to change your program to use these new constants instead of those without the "_L" suffix. For more information see the documentation of `Fl_Preferences`.
1.20.3 Fl::add_timeout and friends

Since FLTK 1.4.0 internal timeout handling has been unified across platforms. This ensures equal timeout handling, improved accuracy of Fl::repeat_timeout(), and easier maintenance (less potential for errors).

This will very likely not affect user code, however there is one subtle exception on macOS and Windows: in FLTK 1.3.x these platforms used system timers to schedule timeouts. Since FLTK 1.4.0 all platforms use the same internal timer management that was previously only used on Unix/Linux/X11. The consequence of this change is that the FLTK event loop needs to be executed to trigger timeout events, i.e. you must either call Fl::wait() repeatedly or start the event loop with Fl::run().

Code that did not execute the event loop and relied on the system timers has never been cross platform compatible, i.e. it wouldn't work on Unix/Linux. An example would be code that opened a splash window, scheduled a timeout with Fl::add_timeout(), and waited for the timer event w/o running the FLTK event loop. Such code must be modified to execute Fl::run() and/or use Fl::wait().

1.20.4 New FL_OVERRIDE Macro

FLTK 1.4 defines a new macro FL_OVERRIDE as "override" if a recent C++ standard (C++11 or higher) is used to compile your code.

This macro is currently defined in FL/Fl_attr.h but this may change in a future release. It is enough to `#include <FL/Fl.H>` to enable this macro.

Unfortunately Visual Studio does not define a meaningful value of __cplusplus to detect the C++ standard. Hence we use the Visual Studio version (2015 or higher) to decide whether we can define FL_OVERRIDE or not.

The FL_OVERRIDE macro is used to decorate declarations of overridden virtual methods in subclasses. Example code from FL/Fl_Window.H:

```
int handle(int) FL_OVERRIDE;
void resize(int X, int Y, int W, int H) FL_OVERRIDE;
Fl_Window * as_window() FL_OVERRIDE { return this; }
```

The FL_OVERRIDE macro translates to 'override' on newer compilers and to an empty string for older compilers.

We recommend to add this to your overridden virtual methods in subclasses derived from FLTK base classes (widgets) and to compile with C++ standard C++11 or higher to enable the compiler to detect some errors if methods are not overridden correctly.

You don't need to declare the overridden methods 'virtual' if you use FL_OVERRIDE or the keyword override.

Hint: For the GCC and clang compilers you can enable the warning '-Wsuggest-override' to detect where you may (want to) add the FL_OVERRIDE macro.

1.20.5 Fl::Image::copy() 'const'

Since FLTK 1.4.0 the virtual method Fl::Image::copy() has been declared 'const' so read-only ('const') images can be copied w/o casts.

This will very likely not affect user code. However, if you derived your own class from any of the Fl_∗_Image variants and you override Your_Image::copy() then you must declare this 'const' as well, i.e. you must add the keyword 'const' to the declaration of copy() in your header file and in the implementation.

We suggest to add the new FL_OVERRIDE macro or the keyword 'override' (see above) to your own overridden method declarations to enable the compiler to detect such incompatibilities.

Code example in header file:
```
class Your_Image {
    // ...
    Fl_Image *copy() const FL_OVERRIDE;
    Fl_Image *copy(int w, int h) const FL_OVERRIDE;
};
```

Note the 'const' attribute and the FL_OVERRIDE macro.
1.21 Software License

December 11, 2001

The FLTK library and included programs are provided under the terms of the GNU Library General Public License (LGPL) with the following exceptions:

1. Modifications to the FLTK configure script, config header file, and makefiles by themselves to support a specific platform do not constitute a modified or derivative work.

   The authors do request that such modifications be contributed to the FLTK project - send all contributions through the "Software Trouble Report" on the following page:  https://www.fltk.org/bugs.php

2. Widgets that are subclassed from FLTK widgets do not constitute a derivative work.

3. Static linking of applications and widgets to the FLTK library does not constitute a derivative work and does not require the author to provide source code for the application or widget, use the shared FLTK libraries, or link their applications or widgets against a user-supplied version of FLTK.

   If you link the application or widget to a modified version of FLTK, then the changes to FLTK must be provided under the terms of the LGPL in sections 1, 2, and 4.

4. You do not have to provide a copy of the FLTK license with programs that are linked to the FLTK library, nor do you have to identify the FLTK license in your program or documentation as required by section 6 of the LGPL.

   However, programs must still identify their use of FLTK. The following example statement can be included in user documentation to satisfy this requirement:

   [program/widget] is based in part on the work of the FLTK project ( https://www.fltk.org ).

GNU LIBRARY GENERAL PUBLIC LICENSE

Version 2, June 1991
Copyright (C) 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.
[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.
This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.
When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".) "Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete
source code means all the source code for all modules it contains, plus any associated interface definition files, plus
the scripts used to control compilation and installation of the library.
Activities other than copying, distribution and modification are not covered by this License; they are outside its
scope. The act of running a program using the Library is not restricted, and output from such a program is covered
only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing
it). Whether that is true depends on what the Library does and what the program that uses the Library does.
1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any
medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice
and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty;
and distribute a copy of this License along with the Library.
You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection
in exchange for a fee.
2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library,
and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet
all of these conditions:
a) The modified work must itself be a software library.
b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of
any change.
c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this
License.
d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program
that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good
faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates,
and performs whatever part of its purpose remains meaningful.
(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent
of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this
function must be optional: if the application does not supply it, the square root function must still compute square
roots.)
These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from
the Library, and can be reasonably considered independent and separate works in themselves, then this License,
and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute
the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on
the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and
every part regardless of who wrote it.
Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather,
the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.
In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on
the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this
License.
3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given
copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the
ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the
ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not
make any other change in these notices.
Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License
applies to all subsequent copies and derivative works made from that copy.
This option is useful when you wish to copy part of the code of the Library into a program that is not a library.
4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or
executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete
corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2
above on a medium customarily used for software interchange.
If distribution of object code is made by offering access to copy from a designated place, then offering equivalent
access to copy the source code from the same place satisfies the requirement to distribute the source code, even
though third parties are not compelled to copy the source along with the object code.
5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by
being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative
work of the Library, and therefore falls outside the scope of this License.
However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the
Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy. For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives
a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients’ exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
1.22 Example Source Code

The FLTK distribution contains over 60 sample applications written in, or ported to, FLTK.

If the FLTK archive you received does not contain either an 'examples' or 'test' directory, you can download the complete FLTK distribution from https://www.fltk.org/software.php.

Most of the example programs were created while testing a group of widgets. They are not meant to be great achievements in clean C++ programming, but merely a test platform to verify the functionality of the FLTK library. Note that extra example programs are also available in an additional 'examples' directory, but these are NOT built automatically when you build FLTK, unlike those in the 'test' directory shown below.

1.22.1 Example Applications: Overview

<table>
<thead>
<tr>
<th>adjuster</th>
<th>animated</th>
<th>arc</th>
<th>ask</th>
<th>bitmap</th>
<th>blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>boxtype</td>
<td>browser</td>
<td>button</td>
<td>buttons</td>
<td>cairo_test</td>
<td>checkers</td>
</tr>
<tr>
<td>clock</td>
<td>colbrowser</td>
<td>color_chooser</td>
<td>cube</td>
<td>CubeView</td>
<td>cursor</td>
</tr>
<tr>
<td>curve</td>
<td>demo</td>
<td>device</td>
<td>doublebuffer</td>
<td>editor</td>
<td>fast_slow</td>
</tr>
<tr>
<td>file_chooser</td>
<td>fluid</td>
<td>fonts</td>
<td>forms</td>
<td>fractals</td>
<td>fullscreen</td>
</tr>
<tr>
<td>gl_overlay</td>
<td>glpuzzle</td>
<td>hello</td>
<td>help_dialog</td>
<td>fractals</td>
<td>fullscreen</td>
</tr>
<tr>
<td>image</td>
<td>inactive</td>
<td>input</td>
<td>input_choice</td>
<td>keyboard</td>
<td>label</td>
</tr>
<tr>
<td>line_style</td>
<td>list_visuals</td>
<td>mandelbrot</td>
<td>menubar</td>
<td>message</td>
<td>minimum</td>
</tr>
<tr>
<td>native-filechooser</td>
<td>navigation</td>
<td>offscreen</td>
<td>output</td>
<td>overlay</td>
<td>pack</td>
</tr>
<tr>
<td>pixmap</td>
<td>pixmap_browser</td>
<td>preferences</td>
<td>radio</td>
<td>resize</td>
<td>resizebox</td>
</tr>
<tr>
<td>rotated_text</td>
<td>scroll</td>
<td>shape</td>
<td>subwindow</td>
<td>sudoku</td>
<td>symbols</td>
</tr>
<tr>
<td>table</td>
<td>tabs</td>
<td>threads</td>
<td>tile</td>
<td>tiled_image</td>
<td>tree</td>
</tr>
<tr>
<td>twowin</td>
<td>unittests</td>
<td>utf8</td>
<td>valuators</td>
<td>windowfocus</td>
<td></td>
</tr>
</tbody>
</table>

1.22.1.1 adjuster

adjuster shows a nifty little widget for quickly setting values in a great range.

1.22.1.2 animated

animated shows a window with an animated square that shows drawing with transparency (alpha channel).

1.22.1.3 arc

The arc demo explains how to derive your own widget to generate some custom drawings. The sample drawings use the matrix based arc drawing for some fun effects.

1.22.1.4 ask

ask shows some of FLTK's standard dialog boxes. Click the correct answers or you may end up in a loop, or you may end up in a loop, or you... .

1.22.1.5 bitmap

This simple test shows the use of a single color bitmap as a label for a box widget. Bitmaps are stored in the X11 `.bmp' file format and can be part of the source code.

Generated by Doxygen
1.22.1.6  blocks

A wonderful and addictive game that shows the usage of FLTK timers, graphics, and how to implement sound on all platforms. blocks is also a good example for the Mac OS X specific bundle format.

1.22.1.7  boxtype

boxtype gives an overview of readily available boxes and frames in FLTK. More types can be added by the application programmer. When using themes, FLTK shuffles boxtypes around to give your program a new look.

1.22.1.8  browser

browser shows the capabilities of the Fl_Browser widget. Important features tested are loading of files, line formatting, and correct positioning of the browser data window.

1.22.1.9  button

The button test is a simple demo of push-buttons and callbacks.

1.22.1.10  buttons

buttons shows a sample of FLTK button types.

1.22.1.11  cairo_test

cairo_test shows a sample of drawing with Cairo in an Fl_Cairo_Window. This program can only be built if FLTK was configured with Cairo support.

1.22.1.12  checkers

Written by Steve Poulsen in early 1979, checkers shows how to convert a VT100 text-terminal based program into a neat application with a graphical UI. Check out the code that drags the pieces, and how the pieces are drawn by layering. Then tell me how to beat the computer at Checkers.

1.22.1.13  clock

The clock demo shows two analog clocks. The innards of the Fl_Clock widget are pretty interesting, explaining the use of timeouts and matrix based drawing.

1.22.1.14  colbrowser

colbrowser runs only on X11 systems. It reads /usr/lib/X11/rgb.txt to show the color representation of every text entry in the file. This is beautiful, but only moderately useful unless your UI is written in Motif.

1.22.1.15  color_chooser

The color_chooser gives a short demo of FLTK's palette based color chooser and of the RGB based color wheel.
1.22.1.16  cube

The cube demo shows the speed of OpenGL. It also tests the ability to render two OpenGL buffers into a single window, and shows OpenGL text.

1.22.1.17  CubeView

CubeView shows how to create a UI containing OpenGL with Fluid.

1.22.1.18  cursor

The cursor demo shows all mouse cursor shapes that come standard with FLTK. The fgcolor and bgcolor sliders work only on few systems (some version of Irix for example).

1.22.1.19  curve

curve draws a nice Bézier curve into a custom widget. The points option for splines is not supported on all platforms.

1.22.1.20  demo

This tool allows quick access to all programs in the test directory. demo is based on the visuals of the IrixGL demo program. The menu tree can be changed by editing test/demo.menu.

1.22.1.21  device

Exercises the Fl_Image_Surface, Fl_Copy_Surface, and Fl_Printer classes to draw to an Fl_Image object, copy graphical data to the clipboard, and for print support.

Note

The clipboard.cxx program of the 'examples' directory is a clipboard watching application that continuously displays the textual or graphical content of the system clipboard (a.k.a pasteboard on Mac OS X) exercising Fl::paste().

1.22.1.22  doublebuffer

The doublebuffer demo shows the difference between a single buffered window, which may flicker during a slow redraw, and a double buffered window, which never flickers, but uses twice the amount of RAM. Some modern OS's double buffer all windows automatically to allow transparency and shadows on the desktop. FLTK is smart enough to not triple buffer a window in that case.

1.22.1.23  editor

FLTK has two very different text input widgets. Fl_Input and derived classes are rather light weight, however Fl_Text_Editor is a complete port of nedit (with permission). The editor test is almost a full application, showing custom syntax highlighting and dialog creation.

1.22.1.24  fast_slow

fast_slow shows how an application can use the Fl_Widget::when() setting to receive different kinds of callbacks.
1.22 Example Source Code

1.22.1.25 file_chooser

The standard FLTK file_chooser is the result of many iterations, trying to find a middle ground between a complex browser and a fast light implementation.

1.22.1.26 fonts

fonts shows all available text fonts on the host system. If your machine still has some pixmap based fonts, the supported sizes will be shown in bold face. Only the first 256 fonts will be listed.

1.22.1.27 forms

forms is an XForms program with very few changes. Search for "ftk" to find all changes necessary to port to ftk. This demo shows the different boxtypes. Note that some boxtypes are not appropriate for some objects.

1.22.1.28 fractals

fractals shows how to mix OpenGL, Glut and FLTK code. FLTK supports a rather large subset of Glut, so that many Glut applications compile just fine.

1.22.1.29 fullscreen

This demo shows how to do many of the window manipulations that are popular for games. You can toggle the border on/off, switch between single- and double-buffered rendering, and take over the entire screen. More information in the source code.

1.22.1.30 gl_overlay

gl_overlay shows OpenGL overlay plane rendering. If no hardware overlay plane is available, FLTK will simulate it for you.

1.22.1.31 glpuzzle

The glpuzzle test shows how most Glut source code compiles easily under FLTK.

1.22.1.32 hello

hello: Hello, World. Need I say more? Well, maybe. This tiny demo shows how little is needed to get a functioning application running with FLTK. Quite impressive, I'd say.

1.22.1.33 help_dialog

help_dialog displays the built-in FLTK help browser. The Fl_Help_Dialog understands a subset of html and renders various image formats. This widget makes it easy to provide help pages to the user without depending on the operating system's html browser.

1.22.1.34 icon

icon demonstrates how an application icon can be set from an image. This icon should be displayed in the window bar (label), in the task bar, and in the task switcher (Windows: Alt-Tab). This feature is platform specific, hence it is possible that you can't see the icon. On Unix/Linux (X11) this can even depend on the Window Manager (WM).
1.22.1.35 iconize

_iconize_ demonstrates the effect of the window functions hide(), iconize(), and show().

1.22.1.36 image

The _image_ demo shows how an image can be created on the fly. This generated image contains an alpha (transparency) channel which lets previous renderings 'shine through', either via true transparency or by using screen door transparency (pixelation).

1.22.1.37 inactive

_inactive_ tests the correct rendering of inactive widgets. To see the inactive version of images, you can check out the pixmap or image test.

1.22.1.38 input

This tool shows and tests different types of text input fields based on _Fl_Input_. The _input_ program also tests various settings of _Fl_Input::when().

1.22.1.39 input_choice

_input_choice_ tests the latest addition to FLTK1, a text input field with an attached pulldown menu. Windows users will recognize similarities to the 'ComboBox'. _input_choice_ starts up in 'plastic' scheme, but the traditional scheme is also supported.

1.22.1.40 keyboard

FLTK unifies keyboard events for all platforms. The _keyboard_ test can be used to check the return values of _Fl::event_key()_ and _Fl::event_text()_. It is also great to see the modifier buttons and the scroll wheel at work. Quit this application by closing the window. The ESC key will not work.

1.22.1.41 label

Every FLTK widget can have a label attached to it. The _label_ demo shows alignment, clipping, and wrapping of text labels. Labels can contain symbols at the start and end of the text, like _@FLTK_ or _@circle uh-huh @square_.

1.22.1.42 line_style

Advanced line drawing can be tested with _line_style_. Not all platforms support all line styles.

1.22.1.43 list_visuals

This little app finds all available pixel formats for the current X11 screen. But since you are now an FLTK user, you don’t have to worry about any of this.

1.22.1.44 mandelbrot

_mandelbrot_ shows two advanced topics in one test. It creates grayscale images on the fly, updating them via the _idle_ callback system. This is one of the few occasions where the _idle_ callback is very useful by giving all available processor time to the application without blocking the UI or other apps.
1.22 Example Source Code

1.22.1.45 menubar

The menubar tests many aspects of FLTK's popup menu system. Among the features are radio buttons, menus taller than the screen, arbitrary sub menu depth, and global shortcuts.

1.22.1.46 message

message pops up a few of FLTK's standard message boxes.

1.22.1.47 minimum

The minimum test program verifies that the update regions are set correctly. In a real life application, the trail would be avoided by choosing a smaller label or by setting label clipping differently.

1.22.1.48 native-filechooser

The native-filechooser program invokes the platform specific file chooser, if available (see Fl_Native_File_Chooser widget).

1.22.1.49 navigation

navigation demonstrates how the text cursor moves from text field to text field when using the arrow keys, tab, and shift-tab.

1.22.1.50 offscreen

offscreen shows how to draw into an offscreen image and display the offscreen image in the program window.

1.22.1.51 output

output shows the difference between the single line and multi line mode of the Fl_Output widget. Fonts can be selected from the FLTK standard list of fonts.

1.22.1.52 overlay

The overlay test app shows how easy an FLTK window can be layered to display cursor and manipulator style elements. This example derives a new class from Fl_Overlay_Window and provides a new function to draw custom overlays.

1.22.1.53 pack

The pack test program demonstrates the resizing and repositioning of children of the Fl_Pack group. Putting an Fl_Pack into an Fl_Scroll is a useful way to create a browser for large sets of data.

1.22.1.54 pixmap

This simple test shows the use of a LUT based pixmap as a label for a box widget. Pixmaps are stored in the X11 `.xpm' file format and can be part of the source code. Pixmaps support one transparent color.
1.22.1.55  pixmap_browser

 pixmap_browser  tests the shared-image interface. When using the same image multiple times, Fl_Shared_Image will keep it only once in memory.

1.22.1.56  preferences

 I do have my preferences in the morning, but sometimes I just can't remember a thing. This is where the Fl_Preferences come in handy. They remember any kind of data between program launches.

1.22.1.57  radio

 The radio tool was created entirely with fluid. It shows some of the available button types and tests radio button behavior.

1.22.1.58  resizebox

 resizebox  shows some possible ways of FLTK's automatic resize behavior.

1.22.1.59  rotated_text

 rotated_text  shows how text can be rotated, i.e. drawn in any given angle. This demo is device specific, for instance it works under X11 only if configured with Xft.

1.22.1.60  resize

 The resize demo tests size and position functions with the given window manager.

1.22.1.61  scroll

 scroll  shows how to scroll an area of widgets, one of them being a slow custom drawing. Fl_Scroll uses clipping and smart window area copying to improve redraw speed. The buttons at the bottom of the window control decoration rendering and updates.

1.22.1.62  shape

 shape  is a very minimal demo that shows how to create your own OpenGL rendering widget. Now that you know that, go ahead and write that flight simulator you always dreamt of.

1.22.1.63  subwindow

 The subwindow demo tests messaging and drawing between the main window and 'true' sub windows. A sub window is different to a group by resetting the FLTK coordinate system to 0, 0 in the top left corner. On Win32 and X11, subwindows have their own operating system specific handle.

1.22.1.64  sudoku

 Another highly addictive game - don't play it, I warned you. The implementation shows how to create application icons, how to deal with OS specifics, and how to generate sound.
1.22.1.65 symbols

symbols are a speciality of FLTK. These little vector drawings can be integrated into labels. They scale and rotate, and with a little patience, you can define your own. The rotation number refers to 45 degree rotations if you were looking at a numeric keypad (2 is down, 6 is right, etc.).

1.22.1.66 table

The table demo shows the features of the Fl_Table widget.

1.22.1.67 tabs

The tabs tool was created with fluid. It tests correct hiding and redisplaying of tabs, navigation across tabs, resize behavior, and no unneeded redrawing of invisible widgets.

The tabs application shows the Fl_Tabs widget on the left and the Fl_Wizard widget on the right side for direct comparison of these two panel management widgets.

1.22.1.68 threads

FLTK can be used in a multithreading environment. There are some limitations, mostly due to the underlying operating system. threads shows how to use Fl::lock(), Fl::unlock(), and Fl::awake() in secondary threads to keep FLTK happy. Although locking works on all platforms, this demo is not available on every machine.

1.22.1.69 tile

The tile tool shows a nice way of using Fl_Tile. To test correct resizing of subwindows, the widget for region 1 is created from an Fl_Window class.

1.22.1.70 tiled_image

The tiled_image demo uses a small image as the background for a window by repeating it over the full size of the widget. The window is resizable and shows how the image gets repeated.

1.22.1.71 tree

The tree demo shows the features of the Fl_Tree widget.

1.22.1.72 twowin

The twowin program tests focus transfer from one window to another window.

1.22.1.73 unitests

unitests exercises all of FLTK’s drawing features (e.g., text, lines, circles, images), as well as scrollbars and schemes.

1.22.1.74 utf8

utf8 shows all fonts available to the platform that runs it, and how each font draws each of the Unicode code points ranging between U+0020 and U+FFFF.
1.22.1.75 valuators

valuators shows all of FLTK's nifty widgets to change numeric values.

1.22.1.76 windowfocus

windowfocus shows a very special case when a new window is shown while the focus stays in the original window.

1.22.1.77 fluid

fluid is not only a big test program, but also a very useful visual UI designer. Many parts of fluid were created using fluid. See the Fluid Tutorial for more details.

1.22.2 Example Applications: Images

This chapter contains a few selected images of the test and example applications listed above. It is not meant to be complete or a full reference. The reason some images are included here is to show how the display should look when running the example programs.

1.22.2.1 cairo_test

The cairo_test demo program shows three shiny buttons drawn with Cairo in an Fl_Cairo_Window.

![Figure 1.43 Buttons drawn with Cairo](image)

1.22.2.2 icon
The **icon** program lets you set the program icon from an image (here an **FL_RGB_Image**).

![Green icon](image1.png)

**Figure 1.44 Green icon (Windows 10)**

### 1.22.2.3 unittests

Select "drawing images" in the browser at the left side to see the image drawing example:

![Image drawing example](image2.png)

**Figure 1.45 Image Drawing**

### 1.23 FAQ (Frequently Asked Questions)

A list of frequently asked questions about FLTK. This appendix describes various frequently asked questions regarding FLTK.

- Where do I start learning FLTK?
- How do I make a box with text?
- Can I use FLTK to make closed-source commercial applications?
- Hitting the 'Escape' key closes windows - how do I prevent this?
1.23.1 Where do I start learning FLTK?

It is assumed you know C++, which is the language all FLTK programs are written in, including FLTK itself. If you like reading manuals to work your way into things, a good start is the FLTK documentation's Introduction to FLTK. Under the FLTK Basics section there's an example 'hello world' program that includes a line-by-line description. If you like looking at simple code first to pique your interest, and then read up from there, start with the example programs in the test/ and examples/ directory that is included with the source code. A good place to start is the 'hello world' program in test/hello.cxx. Also do a google search for "FLTK example programs". "Erco's Cheat Page" is one that shows many simple examples of how to do specific things. If you like to run example programs and look for ones that are like yours and then read them, download and build FLTK from the source, then run the test/demo program. Also, go into the 'examples/' directory and run 'make', then run some of those programs. If you prefer watching TV to reading books and code, google search for "FLTK video tutorials" which has some introductory examples of how to write FLTK programs in C++ and build them.

1.23.2 How do I make a box with text?

The 'hello world' program shows how to make a box with text. All widgets have labels, so picking a simple widget like Fl_Box and setting its label() and using align() to align the label and labelfont() to set the font, and labelsize() to set the size, you can get text just how you want. Labels are not selectable though; if you want selectable text, you can use Fl_Output or Fl_Multiline_Output for simple text that doesn't include scrollbars. For more complex text that might want scrollbars and multiple colors/fonts, use either Fl_Text_Display which handles plain text, or Fl_Help_View which handles simple HTML formatted text.

1.23.3 Can I use FLTK to make closed-source commercial applications?

Yes. The FLTK Software License is standard LGPL, but also includes a special clause ("exception") to allow for static linking. Specifically:

3. Static linking of applications and widgets to the FLTK library does not constitute a derivative work and does not require the author to provide source code for the application or widget, use the shared FLTK libraries, or link their applications or widgets against a user-supplied version of FLTK.

If you link the application or widget to a modified version of FLTK, then the changes to FLTK must be provided under the terms of the LGPL in sections 1, 2, and 4.

4. You do not have to provide a copy of the FLTK license with programs that are linked to the FLTK library, nor do you have to identify the FLTK license in your program or documentation as required by section 6 of the LGPL.

However, programs must still identify their use of FLTK. The following example statement can be included in user documentation to satisfy this requirement:

[program/widget] is based in part on the work of the FLTK project (https://www.fltk.org).

1.23.4 Hitting the 'Escape' key closes windows - how do I prevent this?

[From FLTK article #378]

1. FLTK has a "global event handler" that makes Escape try to close the window, the same as clicking the close box. To disable this everywhere you can install your own that pretends it wants the escape key and thus stops the default one from seeing it (this may not be what you want, see below about the callbacks):

```c
static int my_handler(int event) {
    if (event == FL_SHORTCUT) return 1; // eat all shortcut keys
    return 0;
}
```
1. Attempts to close a window (both clicking the close box or typing Escape) call that window's callback. The default version of the callback does hide(). To make the window not close or otherwise do something different you replace the callback. To make the main window exit the program:

```c
void my_callback(Fl_Widget*, void*) {
    exit(0);
}
main_window->callback(my_callback);
```

If you don't want Escape to close the main window and exit you can check for and ignore it. This is better than replacing the global handler because Escape will still close pop-up windows:

```c
if (Fl::event() == FL_SHORTCUT && Fl::event_key() == FL_Escape)
    return; // ignore Escape
exit(0);
```

The reason for calling a window callback can also be found using using the `Fl::callback_reason()` method:

```c
if (Fl::callback_reason() == FL_REASON_CANCELLED)
    return; // ignore that the user pressed the Escape key
if (Fl::callback_reason() == FL_REASON_CLOSED)
    save_and_exit(); // user clicked the Close button in the window decoration
exit(0); // fallback for other callback reasons
```

It is very common to ask for confirmation before exiting, this can be done with:

```c
if (fl_choice("Are you sure you want to quit?",
    "continue", "quit", NULL))
    exit(0);
```
Chapter 2

FI_Terminal Technical Documentation

This chapter covers the vt100/xterm style "escape codes" used by FI_Terminal for cursor positioning, text colors, and other display screen control features such as full or partial screen clearing, up/down scrolling, character insert/delete, etc.

2.1 The Escape Codes FI_Terminal Supports

These are the escape codes FI_Terminal actually supports, and is not the 'complete' list that e.g. xterm supports. Most of the important stuff has been implemented, but esoteric features (such as scroll regions) has not.

Features will be added as the widget matures.

--- The CSI (Control Sequence Introducer, or "ESC[") ---

<table>
<thead>
<tr>
<th>CSI Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0 - ICH</td>
<td>Insert blank Chars (default=1)</td>
</tr>
<tr>
<td>[A - CUU</td>
<td>Cursor Up, no scroll/wrap</td>
</tr>
<tr>
<td>[B - CUD</td>
<td>Cursor Down, no scroll/wrap</td>
</tr>
<tr>
<td>[C - CUF</td>
<td>Cursor Forward, no wrap</td>
</tr>
<tr>
<td>[D - CUB</td>
<td>Cursor Back, no wrap</td>
</tr>
<tr>
<td>[#E - CNL</td>
<td>Cursor Next Line (crlf) xterm, 'gnome</td>
</tr>
<tr>
<td>[#F - CPL</td>
<td>Cursor Preceding Line; move to sol and up # lines</td>
</tr>
<tr>
<td>[#G - CHA</td>
<td>Cursor Horizontal Absolute positioning</td>
</tr>
</tbody>
</table>

  - ESC[G - move to column 1 (start of line, sol)
  - ESC[#G - move to column #
  - ESC[#H - (CUP) Cursor Position (#'s are 1 based)

  - ESC[H - go to row #
  - ESC[#H - go to (row #) (default=1)
  - ESC[;#H - go to (row #) (col #)
  - ESC[I - (CHT) Cursor Horizontal Tab: tab forward

  - ESC[;I - tab # times (default 1)
  - ESC[#J - (ED) Erase in Display
  - ESC[OJ - clear to end of display (default)
  - ESC[J] - clear to start of display
  - ESC[J] - clear all lines
  - ESC[J] - clear screen history
  - ESC[#K - (EL) Erase in line

  - ESC[#K - clear to end of line (default)
  - ESC[K] - clear to start of line
  - ESC[K] - clear current line
  - ESC[#L - (IL) Insert # Lines (default=1)
  - ESC[#M - (DL) Delete # Lines (default=1)
  - ESC[#P - (DCH) Delete # Chars (default=1)
  - ESC[#S - (SU) Scroll Up # lines (default=1)
  - ESC[#T - (SD) Scroll Down # lines (default=1)
  - ESC[#X - (ECH) Erase Characters (default=1)
  - ESC[#Z - (CBT) Cursor Backwards Tab

  - ESC[#Z - backwards tab # times (default=1)
  - ESC[;a - (HPR) move cursor relative [columns] (default=[row, col+1]) (NOT IMPLEMENTED)
  - ESC[;b - (REP) repeat prev graphics char # times (NOT IMPLEMENTED)
  - ESC[#d - (VPA) Line Position Absolute [row] (NOT IMPLEMENTED)
  - ESC[#e - (LPA) Line Position Relative [row] (NOT IMPLEMENTED)
  - ESC[#f - (CUP) cursor position (#’s 1 based), same as ESC[N]
  - ESC[#g - (TBC) Tabulation Clear

  - ESC[0g - Clear tabstop at cursor
  - ESC[3g - Clear all tabstops

Generated by Doxygen
ESC[m - (SGR) Set Graphic Rendition

*** Attribute Enable ***

ESC[0m - reset: normal attribs/default fg/bg color (VT100)
ESC[1m - bold (VT100)
ESC[2m - dim
ESC[3m - italic
ESC[4m - underline (VT100)
ESC[5m - blink (NOT IMPLEMENTED) (VT100)
ESC[6m - (unused)
ESC[7m - inverse (VT100)
ESC[8m - (unused)
ESC[9m - strikeout
ESC[21m - doubly underline (Currently this just does single underline)

*** Attribute Disable ***

ESC[22m - disable bold/dim
ESC[23m - disable italic
ESC[24m - disable underline
ESC[25m - disable blink (NOT IMPLEMENTED)
ESC[26m - (unused)
ESC[27m - disable inverse
ESC[28m - disable hidden
ESC[29m - disable strikeout

*** Foreground Text "8 Color" ***

ESC[30m - fg Black
ESC[31m - fg Red
ESC[32m - fg Green
ESC[33m - fg Yellow
ESC[34m - fg Blue
ESC[35m - fg Magenta
ESC[36m - fg Cyan
ESC[37m - fg White
ESC[39m - fg default

*** Background Text "8 Color" ***

ESC[40m - bg Black
ESC[41m - bg Red
ESC[42m - bg Green
ESC[43m - bg Yellow
ESC[44m - bg Blue
ESC[45m - bg Magenta
ESC[46m - bg Cyan
ESC[47m - bg White
ESC[49m - bg default

*** Special RGB Color ***

ESC \ [ 38 ; Red ; Grn ; Blu m - where Red, Grn, Blu are decimal (0-255)
ESC[s - save cursor pos (ansi.sys+xterm+gnome, but NOT vt100)
ESC[u - rest cursor pos (ansi.sys+xterm+gnome, but NOT vt100)
ESC[>m - (DECSCA) Set Cursor style (block/line/blink..) (NOT IMPLEMENTED)
ESC[?2;#r - (DECSRAR) Set scroll Region top;bot (NOT IMPLEMENTED)
ESC[f.$t - (DECRAR) (NOT IMPLEMENTED)

------------------------
--- C1 Control Codes ---
------------------------

<ESC>c - (RIS) Reset term to Initial State
<ESC>D - (IND) Index: move cursor down a line, scroll if at bottom
<ESC>E - (NEL) Next Line: basically do a crlf, scroll if at bottom
<ESC>N - (HTS) Horizontal Tab Set: set a tabstop
<ESC>M - (RI) Reverse Index (up w/scroll)

NOTE: Acronyms in parens are Digital Equipment Corporation’s names these VT features.

2.2 Useful Terminal Escape Code Documentation

Useful links for reference:

- https://vt100.net/docs/vt100-ug/chapter3.html
- https://www.xfree86.org/current/ctlseqs.html
- https://www.x.org/docs/xterm/ctlseqs.pdf
- https://gist.github.com/justinmk/a5102f9a0c1810437885a04a07ef0a91

NOTE: Acrophabetic!
2.3 Fl_Terminal Design Document

When I started this project, I identified the key concepts needed to implement Fl_Terminal:

- Draw and manage multiline Unicode text in FLTK
- Allow per-character colors and attributes
- Efficient screen buffer to handle "scrollback history"
- Efficient scrolling with vertical scrollbar for even large screen history
- Mouse selection for copy/paste
- Escape code management to implement VT100 style / ANSI escape codes.

A class was created for each character, since characters can be either ASCII or Utf8 encoded byte sequences. This class is called Utf8Char, and handles the character, its fg and bg color, and any attributes like dim, bold, italic, etc. For managing the screen, after various experiments, I decided a ring buffer was the best way to manage things, the ring split in two:

- 'screen history' which is where lines scrolled off the top are saved
- 'display screen' displayed to the user at all times, and where the cursor lives

Scrolling the display, either by scrollbar or by new text causing the display to scroll up one line, would simply change an 'offset' index# of where in the ring buffer the top of the screen is, automatically moving the top line into the history, all without moving memory around.

In fact the only time screen memory is moved around is during these infrequent operations:

- during scrolling "down"
- character insert/delete operations within a line
- changing the display size
- changing the history size

So a class "RingBuffer" is defined to manage the ring, and accessing its various parts, either as the entire entity ring, just the history, or just the display.

These three concepts, "ring", "history" and "display" are given abbreviated names in the RingBuffer class's API:

```
NOTE: Abbreviations "hist" and "disp"

"history" may be abbreviated as "hist", and "display" as "disp" in both this text and the source code. 4 character names are used so they line up cleanly in the source, e.g.

    ring_rows()    ring_cols()
    hist_rows()    hist_cols()
    disp_rows()    disp_cols()

4 characters
```

These concepts were able to fit into C++ classes:

**Utf8Char**

Each character on the screen is a "Utf8Char" which can manage the UTF-8 encoding of any character as one or more bytes. Also in that class is a byte for an attribute (underline, bold, etc), and two integers for fg/bg color.
RingBuffer

The RingBuffer class keeps track of the buffer itself, a single array of Utf8Chars called "ring_chars" whose width is ring_cols() and whose height is ring_rows().

The "top" part of the ring is the history, whose width is hist_cols() and whose height is hist_rows(). hist_use_rows() is used to define what part of the history is currently in use.

The "bottom" part of the ring is the display, whose width is disp_cols() and whose height is disp_rows().

An index number called "offset" points to where in the ring buffer the top of the ring currently is. This index changes each time the screen is scrolled, and affects both where the top of the display is, and where the top of the history is.

The memory layout of the Utf8Char character array is:

```
ring_chars[]:
```

```
<table>
<thead>
<tr>
<th>History</th>
<th>hist_rows</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><em>v</em></td>
</tr>
<tr>
<td>Display</td>
<td>disp_rows</td>
</tr>
<tr>
<td></td>
<td><em>v</em></td>
</tr>
</tbody>
</table>
```

|<----------------->|

ring_cols  
hist_cols  
disp_cols

So it's basically a single continuous array of Utf8Char instances where any character can generally be accessed by index# using the formula:

```
ring_chars[ (row*ring_cols)+col ]
```

...where 'row' is the desired row, 'col' is the desired column, and 'ring_cols' is how many columns "wide" the buffer is.

The "offset" index affects that formula as an extra row offset, and the resulting index is then clamped within the range of the ring buffer using modulus.

Methods are used to allow direct access to the characters in the buffer that automatically handle the offset and modulus formulas, namely:

```
u8c_ring_row(row,col)  // access the entire ring by row/col
u8c_hist_row(row,col)  // access just the history buffer
u8c_disp_row(row,col)  // access just the display buffer
```

A key concept is the use of the simple 'offset' index integer to allow the starting point of the history and display to be moved around to implement 'text scrolling', such as when crlf at the screen bottom causes a 'scroll up'.

This is simply an "index offset" integer applied to the hist and disp indexes when drawing the display. So after scrolling two lines up, the offset is just increased by 2, redefining where the top of the history and display are, e.g.

```
Offset is 0:

Display

History

Offset now 2:

Display

History

Display
```

This 'offset' trivially implements "text scrolling", avoiding having to physically move memory around. Just the 'offset' changes, the text remains where it is in memory.
This also makes it appear the top line in the display is ‘scrolled up’ into the bottom of the scrollback ‘history'.
If the offset exceeds the size of the ring buffer, it simply wraps around back to the beginning of the buffer with a modulo.
Indexes into the display and history are also modulo their respective rows, e.g.

```
act_ring_index = (hist_rows + disp_row + offset - scrollbar_pos) % ring_rows;
```

This way indexes for ranges can run beyond the bottom of the ring, and automatically wrap around the ring, e.g.

```
> 2
3  Display
4  <-- offset points here
```

```
deep
index
wraps
```

```
0  Display
1  <-- ring_rows points to end of ring
2 :
3 :
4  disp_row(5) -> 4 :.................:
```

The dotted lines show where the display would be if not for the fact it extends beyond the bottom of the ring buffer (due to the current offset), and therefore wraps up to the top of the ring.
So to find a particular row in the display, in this case a 5 line display whose lines lie between 0 and 4, some simple math calculates the row position into the ring:

```
act_ring_index = (hist_rows + offset + disp_row + ring_rows) % ring_rows;
```

An additional bit of math makes sure if a negative result occurs, that negative value works relative to the end of the ring, e.g.

```
if (act_ring_index < 0) act_ring_index = ring_rows + act_ring_index;
```

This guarantees the act_ring_index is within the ring buffer's address space, with all offsets applied.
The math that implements this can be found in the u8c_xxxx_row() methods, where "xxxx" is one of the concept regions "ring", "hist" or "disp":

```
Utf8Char *u8c;
u8c = u8c_ring_row(rrow);  // address within ring, rrow can be 0..(ring_rows-1)
u8c = u8c_hist_row(hrow);  // address within hist, hrow can be 0..(hist_rows-1)
u8c = u8c_disp_row(drow);  // address within disp, drow can be 0..(disp_rows-1)
```

The small bit of math is only involved whenever a new row address is needed, so in a display that's 80x25, to walk all the characters in the screen, the math above would only be called 25 times, once for each row, and each column in the row is just a simple integer offset:

```
for ( int row=0; row<disp_rows(); row++ ) { // walk rows: disp_rows = 25
    Utf8Char *u8c = u8c_disp_row(row);  // get first char in display 'row'
    for ( int col=0; col<disp_cols(); col++ ) { // walk cols: disp_cols = 80
        u8c[col].do_something(); // work with the char at row/col
    }
}
```

So to recap, the concepts here are:

- The ring buffer itself, a linear array that is conceptually split into a 2 dimensional array of rows and columns whose height and width are:

  ```
  ring_rows -- how many rows in the entire ring buffer
  ring_cols -- how many columns in the ring buffer
  nchars   -- total chars in ring, e.g. (ring_rows * ring_cols)
  ```
• The "history" within the ring. For simplicity this is thought of as starting relative to the top of the ring buffer, occupying ring buffer rows:

\[ 0 .. \text{hist}_\text{rows}() - 1 \]

• The "display", or "disp", within the ring, just after the "history". It occupies the ring buffer rows:

\[ \text{hist}_\text{rows}() .. \text{hist}_\text{rows}() + \text{disp}_\text{rows}() - 1 \]

..or similarly:

\[ (\text{hist}_\text{rows}) .. (\text{ring}_\text{rows} - 1) \]

The following convenience methods provide access to the start and end indexes within the ring buffer for each entity:

- Entire ring ring\_srow() – start row index of the ring buffer (always 0)
- ring\_erow() – end row index of the ring buffer
- "history" part of ring hist\_srow() – start row index of the screen history
- hist\_erow() – end row index of the screen history
- "display" part of ring disp\_srow() – start row index of the display
- disp\_erow() – end row index of the display

The values returned by these are as described above. For the hist\_xxx() and disp\_xxx() methods the 'offset' included into the forumula. (For this reason hist\_srow() won't always be zero the way ring\_srow() is, due to the 'offset')

The values returned by these methods can all be passed to the u8c\_ring\_row() function to access the actual character buffer's contents.

• An "offset" used to move the "history" and "display" around within the ring buffer to implement the "text scrolling" concept. The offset is applied when new characters are added to the buffer, and during drawing to find where the display actually is within the ring.

• The "scrollbar", which only is used when redrawing the screen the user sees, and is simply an additional offset to all the above, where a scrollbar value of zero (the scrollbar tab at the bottom) shows the display rows, and as the scrollbar values increase as the user moves the scrollbar tab upwards, +1 per line, this is subtracted from the normal starting index to let the user work their way backwards into the scrollback history. Again, negative numbers wrap around within the ring buffer automatically.

The ring buffer allows new content to simply be appended to the ring buffer, and the index# for the start of the display and start of scrollback history are simply incremented. So the next time the display is "drawn", it starts at a different position in the ring. This makes scrolling content at high speed trivial, without memory moves. It also makes the concept of "scrolling" with the scrollbar simple as well, simply being an extra index offset applied during drawing.

Mouse Selection

Dragging the mouse across the screen should highlight the text, allowing the user to extend the selection either beyond or before the point started. Extending the drag to the top of the screen should automatically 'scroll up' to select more lines in the scrollback history, or below the bottom to do the opposite.

The mouse selection is implemented as a class to keep track of the start/end row/col positions of the selection, and other details such as a flag indicating if a selection has been made, what color the fg/bg text should appear when text is selected, and methods that allow setting and extending the selection, clearing the selection, and "scrolling" the selection, to ensure the row/col indexes adjust correctly to track when the screen or scrollbar is scrolled.

Redraw Timer

Knowing when to redraw is tricky with a terminal, because sometimes high volumes of input will come in asynchronously, so in that case we need to determine when to redraw the screen to show the new content; too quickly will cause the screen to spend more time redrawing itself, preventing new input from being added. Too slowly, the user won't see new information appear in a timely manner.

To solve this, a rate timer is used to prevent too many redraws:

• When new data comes in, a 1/10 sec timer is started and a modify flag is set.
• redraw() is NOT called yet, allowing more data to continue to arrive quickly
- When the 1/10th second timer fires, the callback checks the modify flag:
  - if set, calls redraw(), resets the modify to 0, and calls Fl::repeat_timeout() to repeat the callback in another 1/10th sec.
  - if clear, no new data came in, so DISABLE the timer, done.

In this way, redraws don't happen more than 10x per second, and redraw() is called only when there's new content to see.

The redraw rate can be set by the user application using the Fl_Terminal::redraw_rate(), 0.10 being the default. Some terminal operations necessarily call redraw() directly, such as interactive mouse selection, or during user scrolling the terminal's scrollbar, where it's important there's no delay in what the user sees while interacting directly with the widget.
Chapter 3

The gtk protocol

3.1 Interfaces

- gtk_shell1 - gtk specific extensions
- gtk_surface1 -

3.2 gtk_shell1

3.2.1 Description

gtk_shell is a protocol extension providing additional features for clients implementing it.

3.2.2 API

See The gtk_shell1 interface.

3.3 gtk_surface1

3.3.1 API

See The gtk_surface1 interface.
Chapter 4

Development of the FLTK library

- The Wayland backend for its developer
- Developer info for bundled libs
- Developer Information

4.1 The Wayland backend for its developer

This chapter describes how the Wayland backend of FLTK works from a developer's viewpoint.

4.1.1 Introduction to Wayland

Wayland usage involves communication via a Unix domain socket between a client application and another process called the Wayland compositor which creates, moves, resizes and draws windows on the display. Diverse Wayland compositors exist. They can follow rather diverse logics. For example, FreeBSD offers Sway which is a tiling compositor where the display is always entirely filled with whatever resizable windows are mapped at any given time. Compositors follow either the client-side decoration (CSD) rule where client apps draw window titlebars, or the server-side decoration (SSD) rule where the compositor draws titlebars. FLTK supports both CSD and SSD compositors. It uses a library called libdecor charged of determining whether a CSD or a SSD compositor is active, and of drawing titlebars in the first case.

Wayland is divided in various protocols that a given compositor may or may not support, although they all support the core protocol. Each protocol adds functionality not available in the core protocol. Wayland Explorer lists all protocols. The core protocol allows a client app to discover what protocols the connected compositor supports. Protocols can be stable, which means they have a defined API that will not change but can be expanded, or unstable. For example, mapping a window on a display is not done by the core protocol but by the xdg shell protocol which is stable. The names of symbols used by unstable protocols always begin with letter ‘z’. For example, FLTK uses unstable protocol Text input to support CJK input methods; its symbol names begin with zwp_text_input_v3.

Wayland makes intensive use of the listener mechanism. A listener is a small array of pointers to FLTK-defined callback functions associated to a Wayland-defined object; Wayland calls these functions when defined events occur (more at Listeners below).

Wayland differs noticeably from X11 in that rendering is left to clients: Wayland provides no drawing API. Instead, Wayland provides objects of type struct wl_buffer which encapsulate a memory array of pixel values shared between the client and the compositor. The client app is expected to draw to that memory buffer with whatever means it chooses, and to instruct the compositor to map those pixels to the display when the drawing is complete. The Wayland platform of FLTK draws with the Cairo library to Fl_Window’s and Fl_Image_Surface’s, and with OpenGL to Fl_Gl_Window’s.

Wayland differs also from X11 in that the position of a window in the display is completely hidden to the client app. This prevents function Fl_Window::position() from having any effect on a top-level window. Wayland also prevents a client app from knowing whether a window is minimized: Fl_Window::show() has no effect on a minimized window. Subwindows can be positioned as usual relatively to their parent window. Wayland allows to create popup windows positioned relatively to a previously mapped other window. This allows FLTK to position adequately menu and tooltip windows (see Menu windows and other popups). FLTK uses also popups for the small,
yellow windows that display the new scale factor value when it’s changed: these are created as short-lived popups centered above `Fl::first_window()`.

Wayland uses a trick of its own to handle lists of linked records. It defines type `struct wl_list` and a few macros (`wl_list_init()`, `wl_list_for_each()`, `wl_list_insert()`, `wl_list_for_each_safe()`, `wl_list_remove()`) to manage linked lists. Records put in these lists must contain a member variable of type `struct wl_list` used to link records together and often named ‘link’. Access to such a list is possible memorizing a value of type `struct wl_list` computed by macro `wl_list_init()`. Macro `wl_list_for_each(arg1, arg2, arg3)` allows to run through all list elements with:

- `arg1` is a pointer variable of the type of elements of the linked list;
- `arg2` is the address of a variable of type `struct wl_list` identifying the targeted list;
- `arg3` is the name of the member variable of these elements used to link them together.

For example, `wl_list_for_each()` can be used as follows to scan the linked list of all displays of the system (see `Fl_Wayland_Screen_Driver::output()`):

```c
Fl_Wayland_Screen_Driver *scr_driver = (Fl_Wayland_Screen_Driver*)Fl::screen_driver();
wl_list_for_each(output, &(scr_driver->outputs), link) {
  // ... work with output, an item of the linked list of all displays in the system ...
}
```

Overall, and ignoring for now OpenGL usage, FLTK interacts with Wayland as follows:

- When opening the display: FLTK calls `Fl::add_fd()` in `FL_READ` mode to associate a callback function to the socket connecting the client and the compositor.
- Client to compositor: FLTK calls C functions of the `libwayland-client.so`, `libwayland-cursor.so` and `libxkbcommon.so` shared libraries and of the libdecor library. These send suitable messages to the compositor writing to the socket. The names of these functions begin with `wl_*`, `xkb_*` or `libdecor_*`.
- Compositor to client: the callback function runs when there are data to read in the socket; it calls `wl_*display_dispatch()` which interprets the read data and calls corresponding listeners.

The core protocol defines also a number of mostly opaque structures whose names begin with `wl_*`. The names of symbols and types defined by the other protocols FLTK uses begin with `xdg_*`, `zwp_text_input_v3`, `zxdg_toplevel_decoration`, `gtk_shell1_` and `gtk_surface1_`. FLTK defines a few structures holding Wayland-related data. The names of FLTK-defined structures don’t begin with `wl_*`. For example, `struct wld_window` (see `wld_window`) is used to store all Wayland-specific data associated to a mapped `Fl_Window`.

### 4.1.2 Building libfltk as a Wayland client

Classes `Fl_Wayland_Window_Driver`, `Fl_Wayland_Screen_Driver`, `Fl_Wayland_Graphics_Driver`, `Fl_Wayland_Copy_Surface_Driver`, `Fl_Wayland_Image_Surface_Driver`, `Fl_Wayland_Gl_Window_Driver` and file `fl_wayland_platform_init.cxx` contain all the Wayland-specific code of the FLTK library. This code is located at `src/drivers/Wayland/` in the FLTK source tree. A single C++ source file generally contains all the code of a given class. The code related to copy, paste and drag-and-drop operations, however, is gathered in file `fl_wayland_clipboard_dnd.cxx` and contains a few member functions of class `Fl_Wayland_Screen_Driver`. Furthermore, class `Fl_Unix_System_Driver` is used by both the Wayland and the X11 FLTK platforms. File `FL/fl_config.h` defines preprocessor variables `FLTK_USE_WAYLAND` and `FLTK_USE_CAIRO`.

The public C API to Wayland, xkb, EGL and libdecor libraries are obtained with:

```c
#include <wayland-client.h>
#include <wayland-cursor.h>
#include <xkbcommon/xkbcommon.h>
#include <xkbcommon/xkbcommon-compose.h>
#include <linux/input.h> // for BTN_LEFT, BTN_RIGHT, BTN_MIDDLE
#include "./libdecor/src/libdecor.h"
#include "./libdecor/src/libdecor-plugin.h"
#if HAVE_GL
  # include <wayland-eegl.h>
  # include <EGL/egl.h>
#endif // HAVE_GL
```

as necessary.

File `README.Wayland.txt` details what software packages are needed on Debian-based, Fedora and FreeBSD systems for FLTK to use Wayland. Wayland protocols are packaged as XML files accompanied by a utility program, `wayland-scanner`, able to generate a header file and a necessary glue C source file from a given XML file. For
example, for FLTK to use the XDG shell protocol, these commands are run at build time to generate a .c file that will be compiled into libfltk and a header file that FLTK code will include:

```bash
PROTOCOLS=`pkg-config --variable=pkgdatadir wayland-protocols`
wayland-scanner private-code $PROTOCOLS/stable/xdg-shell/xdg-shell.xml xdg-shell-protocol.c
wayland-scanner client-header $PROTOCOLS/stable/xdg-shell/xdg-shell.xml xdg-shell-client-protocol.h
```

Similar operations are performed for FLTK to use protocols XDG decoration, Text input and GTK Shell.

### 4.1.3 The hybrid Wayland/X11 platform

The Wayland platform of FLTK is normally a two-legged hybrid able to use either Wayland or X11 and to choose between these possibilities at run-time, without any change to the client application. The Wayland/X11 hybrid is essentially a version of the FLTK library containing both all Wayland-specific and all X11-specific code. That’s reflected in file FL/fl_config.h which defines both FLTK_USE_WAYLAND and FLTK_USE_X11. This creates the constraint that Wayland and X11 cannot use the same type name for different purposes or the same symbol name. That is why function fl_xid(const Fl_Window *) is deprecated in FLTK 1.4 and replaced by fl_wl_xid() for Wayland and fl_x11_xid() for X11. Also, global variable Window fl_window is not used by the Wayland platform which instead uses static struct wld_window *Fl_Wayland_Window_Driver:: wld_window.

The FLTK library contains also a short source file, fl_wayland_platform_init.cxx, that determines, at startup time, whether the app will run as a Wayland or as an X11 client. Function attempt_wayland() therein performs this choice as follows:

- if the app defines a global bool variable called fl_disable_wayland and this variable is true, the X11 leg is chosen;
- if environment variable FLTK_BACKEND is defined to string "wayland", the Wayland leg is chosen;
- if environment variable FLTK_BACKEND is defined to string "x11", the X11 leg is chosen;
- otherwise, a connection to a Wayland compositor is attempted; if it’s successful, the Wayland leg is chosen; if it’s not, the X11 leg is chosen.

The first condition listed above is meant to facilitate transition to FLTK 1.4 of source code written for FLTK 1.3 and containing X11-specific code: it’s enough to put

```c
FL_EXPORT bool fl_disable_wayland = true;
```

anywhere in the source code, for the app to run with 1.4, using the x11 leg of the hybrid platform, without any other change in the source code nor to the application’s environment.

In special situations, such as with embedded systems equipped with the Wayland software but lacking the X11 library, it’s possible to build the FLTK library such as it contains only the Wayland backend. This is achieved building FLTK with cmake -DOPTION_WAYLAND_ONLY=on or with configure -disable-x11. In that case, FL/fl_config.h does not define FLTK_USE_X11.

The rest of this chapter describes what happens when the Wayland leg has been chosen.

### 4.1.4 Listeners

A Wayland ‘listener’ is a small array of pointers to FLTK-defined callback functions associated to a Wayland-defined object: Wayland calls these functions when defined events occur, and transmits relevant information to the client app as parameters of these calls. Each listener is associated to its corresponding Wayland object, usually right after the object’s creation, by a call to a specific Wayland function named following the form `wl_XXX_add_listener()`.

For example, this code:

```c
struct wl_surface_listener surface_listener = {
    .surface_enter = surface_enter,
    .surface_leave = surface_leave,
};
```

creates a Wayland object of type `struct wl_surface`, and associates it with a 2-member listener called `surface_listener`. After this, Wayland is expected to call the 2 listener members, `surface_enter` or `surface_leave`, each time my wl_surface will enter or leave, respectively, a display. The arguments of these calls, not detailed here, allow the member functions to identify which surface enters or leaves which display.
The `wl_surface_add_listener()` call above also associates `ptor_to_data` to my wl_surface as user data. The `wl_surface` object's "user data" can be obtained later calling function `wl_surface_get_listener()`.

Wayland function `wl_proxy_get_listener()` returns a pointer to a Wayland object's listener provided that object is transmitted cast to type `struct wl_proxy *`. This gives a handy way to distinguish FLTK-created Wayland objects from objects of other origin: the listener of an FLTK-created object is a known FLTK listener. For example, function `Fl_Wayland_Window_Driver::surface_to_window()` uses this possibility calling `wl_proxy_get_listener( (struct wl_proxy *)wl_surface )` for any object of type `struct wl_surface`: if that object was created as in the example above, this call returns a pointer to FLTK's `surface_listener` static variable.

### 4.1.5 Opening a Wayland connection

Establishing a Wayland connection requires environment variable `XDG_RUNTIME_DIR` to be defined and to point to a directory containing a socket connected to a Wayland compositor. This variable is usually set by the login procedure of Wayland-friendly desktops. The name of the Wayland socket is determined as follows:

- the client may call `Fl::display(const char *display_name)` before `fl_open_display()` runs or use the `-display` command line argument and transmit there the socket name;
- environment variable `NAYLAND_DISPLAY` can be defined to the socket name;
- otherwise, "wayland-0" is used.

What socket is selected determines what compositor will be used by the client application. Function `Fl_Wayland_Screen_Driver::open_display_platform()` establishes the connection to the Wayland socket identified above calling `wl_display_connect(NULL)`, which returns a `struct wl_display *` display pointer or NULL in case of failure. Such NULL return is the hint that allows the FLTK display opening procedure of the Wayland/X11 hybrid to recognize when Wayland access is not possible and to fallback to X11. Then, function `wl_registry_add_listener()` associates a 2-member listener, whose 1st member, `registry_handle_global()`, will be called by Wayland a number of times to indicate each time a protocol supported by the compositor or a system feature such as displays and keyboards. This code allows to run the client until all calls to `registry_handle_global()` have occurred:

```c
static void sync_done(void *data, struct wl_callback *cb, uint32_t time) {
    *(struct wl_callback **)data = NULL;
    wl_callback_destroy(cb);
}

static const struct wl_callback_listener sync_listener = {
    sync_done,
};

struct wl_callback *registry_cb = wl_display_sync(wl_display);
while (registry_cb) wl_display_dispatch(wl_display);
```

A pointer to an object of type `struct wl_callback` created by function `wl_display_sync()` is assigned to variable `registry_cb`. Then a 1-member listener is attached to this object. Wayland will run this listener's member function, `sync_done()`, after all calls to `registry_handle_global()` have occurred. Function `sync_done()` sets to null variable `registry_cb` and destroys the `wl_callback`. Finally, function `wl_display_dispatch()` is called as long as variable `registry_cb` is not null. This makes Wayland process all its pending requests until `sync_done()` runs.

The prototype of function `registry_handle_global()` is:

```c
static void registry_handle_global(void *user_data, struct wl_registry *wl_registry,
                                    const char *interface, uint32_t version)
```

Each time Wayland calls `registry_handle_global()`, `interface` and `version` give the name and version of a component or feature of the Wayland system. It's necessary to call each time function `wl_registry_bind()` which returns a pointer to a Wayland structure that will be the client's access point to the corresponding Wayland protocol or system feature. If that object was created as in the example above, this call returns a pointer to FLTK's `surface_listener` static variable.

Wayland calls `registry_handle_global()` with its parameter `interface` equals to "wl_output" once for each screen connected to the system. Each time, an object of type `struct wl_output` is created, to which a 4-


member listener is associated by function \texttt{wl_output_add_listener()}. The 3rd member of this 4-function listener, \texttt{output\_done()}, runs after all initialization steps of the screen have completed and turns to true member done of a record of type struct \texttt{Fl\_Wayland\_Screen\_Driver::output} associated to the screen. Function \texttt{sync\_done()} mentioned above therefore also calls \texttt{wl\_display\_dispatch()} until the done member of all \texttt{Fl\_Wayland\_Screen\_Driver::output} records are true. Overall, after return from function \texttt{sync\_done()}, FLTK has been made aware of all possible protocols and features of its connected Wayland compositor, and has initialized all screens of the system.

Finally, function \texttt{wl\_display\_get\_fd()} is called to obtain the file descriptor of the Wayland socket and a call to \texttt{Fl::add\_fd()} makes FLTK listen to this descriptor in \texttt{FL\_READ} mode and associates function \texttt{wayland\_socket\_callback()} from file \texttt{Fl\_Wayland\_Screen\_Driver.cxx} with it. This function calls \texttt{wl\_display\_dispatch()} which reads and interprets data available from the file descriptor, and calls corresponding listeners. The \texttt{wl\_display\_dispatch()} call is repeated as long as data are available for reading.

The event loop is run by function \texttt{Fl\_Unix\_System\_Driver::wait()} which is used by both the Wayland and X11 FLTK backends. Among various tasks, this function waits for data on all file descriptors FLTK is listening. Overall, the event loop of the Wayland backend is nearly exactly the same that as used by the X11 backend. The Wayland backend differs only in the callback function handling data read from the Wayland connection socket, and in overridden functions \texttt{Fl\_Wayland\_Screen\_Driver::poll\_or\_select}, \texttt{Fl\_Wayland\_Screen\_Driver::poll\_or\_select\_with\_delay()} and \texttt{Fl\_Wayland\_Screen\_Driver::poll\_or\_select\_with\_delay()}.

\subsection{4.1.6 Wayland windows and surfaces}

Wayland defines objects called surfaces of type struct \texttt{wl\_surface}. A Wayland surface "has a rectangular area which may be displayed on zero or more displays, present buffers, receive user input, and define a local coordinate system". In other words, surface is the name Wayland uses for a window. Buffers allow the client app to draw to surfaces (see Wayland buffers). FLTK creates a surface each time an \texttt{Fl\_Window} is show()’n calling function \texttt{wl\_compositor\_create\_surface()}. Static member function \texttt{Fl\_Wayland\_Window\_Driver::surface\_to\_window(struct \texttt{struct} \texttt{wl\_surface} * \texttt{surface})} gives the \texttt{Fl\_Window} of the surface given as argument. FLTK recognizes 4 distinct kinds of surfaces named DECORATED, UNFRAMED, POPUP and SUBWINDOW. DECORATED are toplevel windows with a titlebar. UNFRAMED have no titlebar. POPUP correspond to menus and tooltips, SUBWINDOW to an Fl\_Window embedded in another Fl\_Window. Function \texttt{Fl\_Wayland\_Window\_Driver::makeWindow()} creates all these surfaces, creates for each a record of type struct \texttt{wld\_window} (see \texttt{wld\_window}), and stores the window kind in member variable kind of this record. Member variable xid of the window’s Fl\_X record stores the adress of this record.

Except for SUBWINDOW’s, each surface needs a Wayland object of type struct \texttt{xdg\_surface} used to make it become a mapped window and stored in member \texttt{xdg\_surface} of the window’s \texttt{wld\_window} record. For DECORATED windows, this object is created inside libdecor and transmitted to FLTK by function \texttt{libdecor\_frame\_get\_xdg\_surface()} for UNFRAMED and POPUP windows, it’s created by function \texttt{xdg\_wm\_base\_get\_xdg\_surface()}. Finally, each surface is also associated to one more Wayland object whose type varies with the window’s kind. These explain this part of the \texttt{wld\_window} record:

\begin{verbatim}
union {
  struct libdecor_frame *frame; // created for DECORATED by libdecor_decorate()
  struct wl_subsurface *subsurface; // created for SUBWINDOW by wl_subcompositor_get_subsurface()
  struct xdg_popup *xdg_popup; // created for POPUP by xdg_surface_get_popup()
  struct xdg_toplevel *xdg_toplevel; // created for UNFRAMED by xdg_surface_get_toplevel()
};
\end{verbatim}

Except for SUBWINDOW’s, each surface is associated to a ‘configure’ function that Wayland calls one or more times when the window is going to be mapped on the display. The ‘configure’ function of DECORATED surfaces is \texttt{handle\_configure()} which is the 1st member of a 4-member listener named \texttt{libdecor\_frame\_configure()} associated to a decorated window when it’s created calling \texttt{libdecor\_decorate()}. Finally, a call to \texttt{libdecor\_frame\_map()} triggers the process of mapping the newly created DECORATED surface on a display. Wayland calls \texttt{handle\_configure()} twice during this process. The first \texttt{handle\_configure()} run allows to set the window’s \texttt{xdg\_surface} object which is returned by function \texttt{libdecor\_frame\_get\_xdg\_surface()} for UNFRAMED and POPUP surfaces, it’s created by function \texttt{xdg\_surface\_configure()} and \texttt{xdg\_toplevel\_configure()} and \texttt{popup\_configure()}. The mapping process of these surfaces is triggered by a call to \texttt{wl\_surface\_commit()}. These ‘configure’ functions transmit effective window size information to FLTK. Also, they are where the window’s \texttt{Fl\_Window\_Driver::wait\_for\_}...
expose_value member variable is set to 0 to indicate that the window has been mapped to display. **Caution**: there are some small differences between how and when the various Wayland compositors call handle_configure().

When a decorated window changes size, whatever the cause of it, Wayland calls handle_configure() which sets member variable Fl_Wayland_Window_Driver::in_handle_configure to true and calls the window's virtual resize() function which ultimately runs Fl_Wayland_Window_Driver::resize() which calls Fl_Group::resize() to perform FLTK's resize operations and Fl_Wayland_Graphics_Driver::buffer_release() to delete the existing window buffer that's not adequate for the new window size. At the end of the run of handle_configure(), in_handle_configure is set back to false. When the window size change is caused by the app itself calling the window's resize() function, Fl_Wayland_Window_Driver::resize() is false. This allows Fl_Wayland_Window_Driver::resize() to detect that Wayland needs be informed of the desired size change, which gets done by a call to libdecor_frame_commit(). Wayland later calls handle_configure() and events described above unfold.

Wayland generally does not provide a way to control where the compositor should map a window in the system displays. Nevertheless, for multi-display systems, Wayland allows to control on what display should the compositor map a fullscreen window. That is done inside function handle_configure() which calls libdecor_frame_set_fullscreen() for DECORATED windows and inside function xdg_toplevel_set_fullscreen() which calls xdg_toplevel_set_fullscreen() for UNFRAMED. The struct wl_output pointer for the targeted display is transmitted as 2nd argument of these calls.

### 4.1.7 Menu windows and other popups

Menu windows, tiny menu title windows, and tooltip windows are implemented using Wayland's popup mechanism which allows to position a popup window relatively to a previously mapped window, itself a popup or another kind of window, with the restriction that any popup must overlap or at least touch that other window. Member function Fl_Wayland_Window_Driver::makeWindow calls member function Fl_Wayland_Window_Driver::process_menu_or_tooltip to create all popups.

This function gets called after FLTK has computed using a given algorithm the desired (x,y) position of the popup window's top-left corner, using coordinates centered on the top-left corner of the toplevel window from which the popup originates. This algorithm is able to prevent popups from being positioned beyond the screen borders under the assumption that the position of a toplevel window inside a screen is known. While this assumption holds for other platforms, it does not for the Wayland platform. The FLTK code for the Wayland platform therefore modifies the algorithm that FLTK uses to compute the position of menu windows. The key information used by this algorithm is obtained by member function Fl_Window::menu_window_area which computes the coordinates of the rectangle where menu windows are allowed to be positioned. Under other platforms, this function just returns the origin and size of the work area of the screen in use. In contrast, the Wayland platform handles two situations differently:

- For menu windows that are not taller than the display in use, the Wayland-overridden member function Fl_Wayland_Window_Driver::menu_window_area returns large negative origin and large width and height values. This lets the standard FLTK algorithm position the menu relatively to its window of origin without concern about screen limits, and relies on Wayland's constraint mechanism described below to prevent the menu from going beyond these limits, without FLTK having to know where they are.

- Menu windows taller than the screen where they are mapped need special handling described in detail in a comment above the source code of function Fl_Wayland_Window_Driver::process_menu_or_tooltip.

Function Fl_Wayland_Window_Driver::process_menu_or_tooltip first computes origin_win, pointer to the Fl_Window relatively to which the popup is to be positioned. Window origin_win is the parent menu window when the popup is a sub-menu; it’s the tiny window title when the popup is a menu with a title; otherwise, it’s the window containing the point of origin of the popup. An object of type struct xdg_positioner created by function xdg_wm_base_create_positioner() is used to express the rules that will determine the popup position relatively to origin_win as follows:

- Function xdg_positioner_set_anchor_rect() determines a rectangle in origin_win relatively to which the popup is to be positioned. When the popup to be created is a menu window spawned by an Fl_Menu_Bar, that rectangle is the full area of the menu title window. Otherwise, that rectangle is an adequately located point.

- Function xdg_positioner_set_size() sets the popup size.
The \texttt{xdg_positioner_set_anchor} function \texttt{(positioner, XDG\_POSITIONER\_ANCHOR\_BOTTOM\_LEFT);} and \texttt{xdg_positioner_set_gravity} \texttt{(positioner, XDG\_POSITIONER\_GRAVITY\_BOTTOM\_RIGHT);} calls position the popup so that its top-left corner is initially below and at right of the bottom-left corner of the
\texttt{origin\_win}’s anchor rectangle.

The call to \texttt{xdg_positioner_set_offset()} further changes the popup vertical position.

The call to \texttt{xdg_positioner_set_constraint_adjustment()} uses constraint flags \texttt{XDG\_POSITIONER\_CONSTRAINT\_ADJUSTMENT\_SLIDE\_X} and \texttt{XDG\_POSITIONER\_CONSTRAINT\_ADJUSTMENT\_SLIDE\_Y} which mean that the compositor will move the popup horizontally and vertically if its initial position would make it expand beyond the edges of the screen. Furthermore, flag \texttt{XDG\_POSITIONER\_CONSTRAINT\_ADJUSTMENT\_SLIDE\_Y} is added when the popup is a menu window spawned by an \texttt{Fl\_Menu\_Bar}; this has the popup flipped above the \texttt{Fl\_Menu\_Bar} if there's not enough screen room below it for the popup.

Finally, a call to function \texttt{xdg\_surface\_get\_popup()} creates the popup accounting for position rules listed above. The positioner is then deleted by \texttt{xdg\_positioner\_destroy()}, a listener is associated to the popup surface with \texttt{xdg\_popup\_add\_listener()}, and a call to \texttt{wl\_surface\_commit()} triggers the mapping of the popup on the display.

Overall, the expected coordinates of the top-left corner of the popup relatively to \texttt{origin\_win} are \texttt{popup\_x, popup\_y}. They are memorized in a record of FLTK-defined type \texttt{struct win\_positioner} that’s associated to the popup listener. When the compositor maps the popup, function \texttt{popup\_configure}, the first element of the popup listener, runs and receives as arguments the coordinates of the popup top left and its size. These values account for the positioning constraints of the popup which may have moved it to avoid screen borders. This function can therefore detect whether constraints applied have modified the effective popup location in comparison to the expected coordinates which are available as member variables of the \texttt{struct win\_positioner} record mentioned above. That's key to the handling by FLTK of full menu windows.

Groups of popups containing a menutitle, the associated menuwindow, and optionally a submenu window and that don’t belong to an \texttt{Fl\_Menu\_Bar} are mapped in a different order: the menuwindow is mapped first, and the menutitle is mapped second above it as a child popup. Function \texttt{Fl\_Window\_Driver::is\_floating\_title()} detects when such a menutitle is created, static member variable \texttt{previous\_floating\_title} is assigned the value of this menutitle, and the menutitle is mapped only after the menuwindow has been mapped, as a child of it. This positions better the popup group in the display relatively to where the popup was created.

### 4.1.8 Fl\_Wayland\_Graphics\_Driver and Fl\_Cairo\_Graphics\_Driver

The Wayland platform of FLTK uses an \texttt{Fl\_Wayland\_Graphics\_Driver} object for all its on-screen drawing operations. This object is created by function \texttt{Fl\_Graphics\_Driver::new\_Main\_Graphics\_Driver()} called by \texttt{Fl\_Display\_Device::display\_device()} when the library opens the display. New \texttt{Fl\_Wayland\_Graphics\_Driver} objects are also created for each \texttt{Fl\_Image\_Surface} and each \texttt{Fl\_Copy\_Surface} used, and deleted when these objects are deleted. Class \texttt{Fl\_Wayland\_Graphics\_Driver} derives from class \texttt{Fl\_Cairo\_Graphics\_Driver} which implements all the FLTK drawing API for a Cairo surface. Function \texttt{Fl\_Wayland\_Graphics\_Driver::init()} creates the Cairo surface used by each \texttt{Fl\_Wayland\_Graphics\_Driver} object by calling \texttt{cairo\_image\_surface\_create\_for\_data()} for the window’s or offscreen’s draw buffer (see below).

Class \texttt{Fl\_Cairo\_Graphics\_Driver} is also used by the X11 leg of the hybrid Wayland-X11 platform because this leg draws to the display with an \texttt{Fl\_X11\_Cairo\_Graphics\_Driver} object which derives from class \texttt{Fl\_Cairo\_Graphics\_Driver}. Finally, \texttt{Fl\_Cairo\_Graphics\_Driver} is also used, in the form of an object from its derived class \texttt{Fl\_PostScript\_Graphics\_Driver}, when the hybrid Wayland-X11 platform draws PostScript, or when the classic X11 platform uses Pango and draws PostScript. This happens when classes \texttt{Fl\_PostScript\_File\_Device} and \texttt{Fl\_Printer} are used.

### 4.1.9 Wayland buffers

Wayland uses buffers, objects of type \texttt{struct wl\_buffer}, to draw to surfaces. In principle, one or more buffers can be associated to a surface, and functions \texttt{wl\_surface\_attach()} and \texttt{wl\_surface\_commit()} are called to first attach one such buffer to the surface and then inform the compositor to map this buffer’s graphics content on the display. Wayland buffers can use various memory layouts. FLTK uses \texttt{WL\_SHM\_FORMAT\_ARGB8888}, which is the same layout as what Cairo calls \texttt{CAIRO\_FORMAT\_ARGB32}.
FLTK calls function `Fl_Wayland_Window_Driver::make_current()` before drawing to any `Fl_Window`. Member buffer of this `Fl_Window`'s struct `wld_window` (see `wld_window`) is NULL when the window has just been created or resized. In that case, FLTK calls `Fl_Wayland_Graphics_Driver::create_wld_buffer()` which returns a pointer to a struct `wld_buffer` containing

- a Wayland buffer, member `wl_buffer`;  
- a Cairo image surface, created by a call to `Fl_Wayland_Graphics_Driver::cairo_init()`.

Each of these two objects encapsulates a byte array of the same size and the same memory layout destined to contain the `Fl_Window`'s graphics. The Cairo image surface object is where FLTK draws. The Wayland buffer is what Wayland maps on the display. FLTK copies the Cairo surface's byte array to the Wayland buffer's byte array before beginning the mapping operation. If `width` and `height` are a window's dimensions in pixels,

```c
int stride = cairo_format_stride_for_width(CAIRO_FORMAT_ARGB32, width);
int size = stride * height;
```

give size, the common size of both byte arrays.

The effective creation of the `wl_buffer` object is delayed until function `Fl_Wayland_Graphics_Driver::buffer_commit()` gets called. Section Buffer factories below details how FLTK creates `wl_buffer` objects.

The `struct Fl_Wayland_Graphics_Driver::wld_buffer` (see `wld_buffer`) contains a pointer to the byte array of the Cairo image surface (member `draw_buffer.buffer`), information about the Wayland buffer (members `wl_buffer` and `data`), the common size of the Cairo surface's and Wayland buffer's byte arrays (member `draw_buffer.data_size`), and other information. A pointer to this struct `Fl_Wayland_Graphics_Driver::wld_buffer` is memorized as `member_buffer` of the `Fl_Window`'s `wld_window`. All drawing operations to the `Fl_Window` then modify the content of the Cairo image surface.

Function `Fl_Wayland_Window_Driver::flush()` is in charge of sending FLTK graphics data to the display. That is done by calling function `Fl_Wayland_Graphics_Driver::buffer_commit()` which creates the `struct wl_buffer` object calling `create_shm_buffer()` if that was not done before, copies the byte array of the Cairo surface to the Wayland buffer's starting memory address, and calls functions `wl_surface_attach()` and `wl_surface_commit()`. Before calling `Fl_Window::flush()`, FLTK has computed a damaged region. If that region is not null, `Fl_Wayland_Graphics_Driver::buffer_commit()` copies only the damaged parts of the Cairo surface to the Wayland buffer and calls function `wl_surface_damage_buffer()` for these parts to inform the compositor of what parts of the surface need its attention.

### Throttling redraw operations

An important detail here is that FLTK uses Wayland's synchronization mechanism to make sure the surface's `wl_buffer` is not changed while the compositor is using it and to refrain from calling `wl_surface_commit()` more frequently than the system can process it. This 2-step mechanism works as follows:

- `Fl_Wayland_Graphics_Driver::buffer_commit()` first calls function `wl_surface_frame()` to obtain a pointer to a `struct wl_callback` object and stores it as member `frame_cb` of the surface's `wl_window`. Then it calls `wl_callback_add_listener()` to associate this object to the FLTK-defined, callback function `surface_frame_done()`. It next calls `wl_surface_commit()`. Together, these 3 calls instruct Wayland to start mapping the buffer content to the display and to call `surface_frame_done()` later, when it will have become ready for another mapping operation.

- Later, `surface_frame_done()` runs and destroys the `wl_callback` object by function `wl_callback_destroy()` and sets member `frame_cb` to `NULL`.

Member variable `draw_buffer_needs_commit` of the `wl_buffer` is also important in this mechanism: it informs FLTK that the graphics buffer has changed and needs being committed. This variable is turned `true` every time a graphics operation changes the buffer content and turned `false` when the buffer gets committed.

This procedure ensures that FLTK never changes the surface’s Wayland buffer while it’s being used by the compositor and never calls `wl_surface_commit()` before Wayland gets ready for a new commit because `Fl_Wayland_Window_Driver::flush()` calls `Fl_Wayland_Graphics_Driver::buffer_commit()` only if `frame_cb` is `NULL`. If it’s not `NULL`, the exact content of function `surface_frame_done()` is:

```c
static void surface_frame_done(void *data, struct wl_callback *cb, uint32_t time) {
  struct wld_window *window = (struct wld_window *)data;
  wl_callback_destroy(cb);
  window->frame_cb = NULL;
  if (window->buffer && window->buffer->draw_buffer_needs_commit) {
    static void surface_frame_done(void *data, struct wl_callback *cb, uint32_t time) {
      struct wld_window *window = (struct wld_window *)data;
      wl_callback_destroy(cb);
      window->frame_cb = NULL;
      if (window->buffer && window->buffer->draw_buffer_needs_commit) {  
```
4.1 The Wayland backend for its developer

has the effect that when the mapping operation eventually completes, Wayland runs `surface_frame_done()`, which calls anew `Fl_Wayland_Graphics_Driver::buffer_commit()` if the buffer's `draw_buffer←_needs_commit` member is true. The net result is that the screen shows the most recent surface content.

This synchronization mechanism is also used when performing an interactive window resize operation. During such operation, the compositor informs the client an interactive resize is being performed and sends window resize commands at high rate (~60 Hz) to the client via the socket. Libdecor turns on flag `LIBDECOR_WINDOW←_STATE_RESIZING` to inform the client, and runs function `handle_configure()` for each received resize command. Before calling `Fl_Group::resize()` and later `Fl_Window::draw()`, `handle_configure()` tests whether `window→frame_cb` is NULL. When it's not because a previous resize operation is being performed, the current resize command is skipped. At the end of the interactive resize, flag `LIBDECOR_WINDOW_STATE_RESIZING` is off and Wayland sends a final resize command which is not skipped. Overall, this ensures the client program resizes its window as frequently as it can without falling behind resize commands sent by the compositor.

To account for a bug in Mutter (issue #878), the `window→frame_cb` object is not created when a toplevel window is being resized and is entirely covered by one subwindow.

Progressive window drawing

FLTK supports progressive drawing when an app calls function `Fl_Window::make_current()` at any time and then calls the FLTK drawing API. This is made possible in function `Fl_Wayland_Window_Driver::make←current()` with

```c
// to support progressive drawing
if ( (!Fl_Wayland_Window_Driver::in_flush_) && window->buffer && (!window->frame_cb) && window->buffer->draw_buffer_needs_commit && (!wait_for_expose_value) ) {
  Fl_Wayland_Graphics_Driver::buffer_commit(window);
}
```

Thus, `buffer_commit()` runs only when `frame_cb` is NULL. If an app rapidly performs calls to `Fl_Window::make_current()` and to drawing functions, FLTK will copy `draw_buffer` to the Wayland buffer and instruct Wayland to map it to the display when `frame_cb` is NULL which means that the compositor is ready to start performing a mapping operation. This occurs when the progressive drawing operation begins. Later, `frame_cb` is generally found non NULL when `Fl_Wayland_Window_Driver::make_current()` runs because the compositor is busy processing the previous Wayland buffer. When the compositor has completed this processing, the client app runs `surface_frame_done()` which, provided member variable `draw_buffer←_needs_commit` is true, calls `Fl_Wayland_Graphics_Driver::buffer_commit()`. This makes the compositor map the Wayland buffer in its new, more advanced, state.

An example of progressive drawing is given by FLTK's mandelbrot test app. When set to fullscreen, this app can be seen to progressively fill its window from top to bottom by blocks of lines, each block appearing when the compositor is ready to map a new buffer. When the compositor is not ready, the app does not block but continues computing and drawing in memory but not on display more lines of the desired Mandelbrot graph.

Wayland buffer deletion

Each `wl_buffer` record contains boolean member `in_use` which is set to `true` just before the buffer gets committed, and boolean member `released` which is set to `true` when FLTK no longer needs the buffer and calls `Fl_Wayland_Graphics_Driver::buffer_release()`. FLTK's buffer-creating function, `Fl_Wayland_Graphics_Driver::create_shm_buffer()`, attaches a 1-member listener to each buffer which Wayland calls after a commit operation to indicate the client is allowed to re-use the buffer. This listener's member function, `buffer_release_listener()`, turns to false member `in_use` of the buffer's `wl_buffer` record.

Since the two events 'FLTK no longer needs the buffer' and 'the client is allowed to re-use the buffer' can arrive in any order, FLTK deletes the `struct wl_buffer` object by running `do_buffer_release()` only after both events happened, that is, when `in_use` is false and `released` is true. That's why function `do_buffer←_release()` is called by both functions `Fl_Wayland_Graphics_Driver::buffer_release()` and `buffer_release_listener()`.

4.1.10 Buffer factories

Wayland names `buffer factory` a software procedure that constructs objects of type `struct wl_buffer` for use by a client application. FLTK creates a `wl_buffer` object each time an `Fl_Window` is mapped on a display or
resized. That’s done by member function Fl_Wayland_Graphics_Driver::create_shm_buffer() which follows this 3-step procedure to create a "buffer factory" for FLTK and to construct Wayland buffers from it:

- Libdecor function os_createAnonymousFile(off_t size) creates an adequate file and mmap’s it. This file lives in RAM because it is created by function memfd_create(). FLTK sets this file size to 10 MB unless the size of the buffer to be created is larger; in that case the anonymous file is sized to twice the buffer size.

- Wayland function wl_shm_create_pool() shares this mmap’ed memory with the Wayland compositor and returns an object of type struct wl_shm_pool which encapsulates this memory. A record of type struct Fl_Wayland_Graphics_Driver::wld_buffer data is created and associated to the newly created wl_shm_pool by wl_shm_pool_set_user_data(). This record stores the starting address (pool_memory) and size (pool_size) of the pool’s encapsulated memory. The record also contains member buffers of type struct wl_list which stores the access point to the linked list of wl_buffer objects that will be created from the wl_shm_pool.

- A variable named chunk_offset represents the offset within the pool’s shared memory available for the buffer being constructed. It equals 0 when the pool has just been created and is updated as detailed below when one or more buffers have been previously created from the pool. A record of type struct Fl_Wayland_Graphics_Driver::wld_buffer is created. This record will contain (member wld_buffer) the address of a wl_buffer object that’s created by function wl_shm_pool_create<buffer>(). This wl_buffer object encapsulates a section of a given size of the pool’s shared memory beginning at offset chunk_offset in it. Quantity pool_memory + chunk_offset is therefore the address of the beginning of the mmap’ed memory section encapsulated by this wl_buffer. Member shm_pool of the newly constructed Fl_Wayland_Graphics_Driver::wld_buffer object is set to the address of the current wl_shm_pool object. This record is added to the head of the linked list of current pool’s buffers by a call to wl_list_insert(). At that point, a struct Fl_Wayland_Graphics_Driver::wld_buffer record part of the linked list of all such records corresponding to wl_buffer objects created from the same wl_shm_pool object, and member shm_pool of this record gives the address of this wl_shm_pool. When a new struct Fl_Wayland_Graphics_Driver::wld_buffer record is to be created,

\[
\text{struct } \text{wld_shm_pool_data} \ast \text{pool_data} =
\text{(struct wld_shm_pool_data *)wl_shm_pool_get_user_data(pool)};
\text{struct Fl_Wayland_Graphics_Driver::wld_buffer } \ast \text{record = wi_container_of(pool_data->buffers.next, record}, \text{link};
\text{int chunk_offset = ((char*)record->data - pool_data->pool_memory) + record->data_size};
\]

gives the offset within the current pool’s mmap’ed memory available for a new wl_buffer. Macro wi_container_of() gives the address of a record belonging to a linked list of records of the same type.

A window’s wl_buffer is re-filled by graphics data and committed each time the window gets redrawn, and is set to be destroyed by function Fl_Wayland_Graphics_Driver::buffer_release() when Fl_Window::hide() runs or the window is resized. When the wl_buffer is no longer in use, function do_buffer_release() gets called as explained above. It destroys the wl_buffer with wl_buffer destroy(), and removes the corresponding Fl_Wayland_Graphics_Driver::wld_buffer record from the linked list of buffers from the same wl_shm_pool. Since new Fl_Wayland_Graphics_Driver::wld_buffer records are added at the head of the linked list, and since the record at the head of this list is used to compute the offset within the pool’s mmap’ed memory available for a new wl_buffer, destruction of the last created wl_buffer allows to re-use the destroyed buffer’s pool’s memory for a new wl_buffer. When function do_buffer_release() finds the list of buffers from a given pool empty, two situations can occur. 1) This pool is the current pool. Its mmap’ed memory will be re-used from offset 0 to create future wl_buffer objects. 2) This pool is not current. It gets destroyed with wl_shm_pool_destroy(), the pool’s mmap’ed memory is munmap’ed, and the pool’s associated struct wl_shm_pool data is freed. In situation 1) above, the next wl_buffer to be created can need more memory than the current pool’s memory size. If so, the current pool gets destroyed and replaced by a new, larger pool.

If the sum of chunk_offset plus the buffer size is larger than the current pool’s size when function create<shm_buffer>() is called, chunk_offset is reset to 0, and a new wl_shm_pool object is created and used by FLTK’s "buffer factory". This mechanism allows to access new mmap’ed memory when chunk_offset reaches the end of the previous mmap’ed section.

Wayland uses also wl_buffer objects to support cursors. FLTK uses the "buffer factory" described here when creating custom cursors (see custom-cursor) with function Fl_Wayland_Window_Driver::set<cursor(const Fl_RGB_Image *, ...) which calls create<shm_buffer>() via set<cursor<4args(), custom_offscreen() and create<wl_buffer>(). In contrast, standard shaped-cursors
(e.g., FL_CURSOR_INSERT) use their own "buffer factory" inside Wayland functions such as `wl_cursor_theme_get_cursor()`. Therefore, the fact that the `wl_buffer` objects behind standard cursors are never destroyed doesn't prevent disused `struct wl_shm_pool` objects from being freed because those buffers come from a distinct "buffer factory". The "buffer factory" described here is also used by function `offscreen_from_text()` when displaying dragged text in a DnD operation.

### 4.1.11 Displays and HighDPI support

Wayland uses the concept of seat of type `struct wl_seat` which encompasses displays, a keyboard, a mouse, and a trackpad. Although Wayland may be in principle able to deal with several seats, FLTK's Wayland platform is conceived for one seat only. That seat may contain one or more displays, which Wayland calls outputs, of type `struct wl_output`.

As written above, function `registry_handle_global()` discovers the available seat at startup time. This function also associates a listener to each display connected to the system by calling function `wl_output_add_listener()`. This listener's member functions run at program startup when Wayland discovers its displays (see Opening a Wayland connection). Member `output_mode` runs also when the display is resized and member `output_scale` also when the Wayland scale factor (see below) is changed. FLTK defines type `struct Fl_Wayland_Screen_Driver::output` to store display size and scaling information. One such record is created for each display. These records are put in a `struct wl_list` accessible from member outputs of the single `Fl_Wayland_Screen_Driver` object.

FLTK uses 2 distinct scaling parameters for each display:

- **`int wld_scale;`**. This member variable of the `struct Fl_Wayland_Screen_Driver::output` record typically equals 1 for standard, and 2 for HighDPI displays. The effect of value `n` of variable `wld_scale` is that `n` Wayland graphics unit represents a block of `nxn` pixels. Another effect is that drawing a buffer for a surface of size `WxH` units contains `W * n * H * n * 4` bytes. Member function `output_scale()` mentioned above sets this value for each system's display at startup time. Member function `Fl_Wayland_Screen_Driver::buffer_commit()` informs the Wayland compositor of the value of `wld_scale` calling `wl_surface_set_buffer_scale()` which is enough to make FLTK apps HighDPI-aware. Under the gnome and KDE desktops, this parameter is visible in the "Settings" app, "Displays" section, "Scale" parameter which is 200% on HighDPI displays.

- **`float gui_scale;`**. This other member variable is where FLTK's own GUI scaling mechanism with `ctrl+//-/0` keystrokes and with environment variable `FLTK_SCALING_FACTOR` operates: when FLTK is scaled at 150%, `gui_scale` is assigned value 1.5. Function `Fl_Wayland_Screen_Driver::scale(int n, float f)` assigns value `f` to the `gui_scale` member variable of display # `n`. This variable is used by function `Fl_Wayland_Window_Driver::make_current()` when it calls `Fl_Wayland_Screen_Driver::set_buffer()` that scales the graphics driver by this factor with `cairo_scale()`.

Overall, an FLTK object, say an `Fl_Window`, of size `WxH` FLTK units occupies `int(W * gui_scale) * wld_scale x int(H * gui_scale) * wld_scale` pixels on the display.

When an `Fl_Window` is to be show()'n, `Fl_Wayland_Window_Driver::makeWindow()` creates a `struct wl_surface` with `wl_compositor_create_surface()` and associates it calling `wl_surface_add_listener()` with a 2-member listener called `surface_listener` charged of managing as follows the list of displays where this `wl_surface` will map. The `Fl_Window` possesses an initially empty linked list of displays accessible at member outputs of the window's `wld_window` record. When the `Fl_Window`, or more exactly its associated `struct wl_surface` is mapped on a display, member `surface_enter()` of `surface_listener` runs. This function adds the display where the surface belongs to the end of the linked list of displays for this surface. When a surface is dragged or enlarged across the edge of a display in a multi-display system and expands on a second display, `surface_enter()` runs again, and this surface's list of displays contains 2 items. When a surface leaves a display, member `surface_leave()` of `surface_listener` runs. It removes that display from the surface's list of displays. Each time the first item of a surface's list of displays changes, function `change_scale()` is called and applies that display's `gui_scale` value to that surface calling `Fl_Window_Driver::screen_num(int)` When a window is unmapped by function `Fl_Wayland_Window_Driver::hide()`, the surface's list of displays is emptied.
Fractional scaling

The KWin compositor, and gnome too if specially set, allow to use fractional scaling that can take intermediate values between 100% and 200%. Wayland implements this rendering all *wl_surface*’s as if the scaling was at 200%, and downsizing them to the desired fractional scale value at the compositing stage. Seen from FLTK, everything runs as when *wld_scale* = 2.

These commands make gnome accept fractional scaling, and turn that off:

- `gsettings set org.gnome.mutter experimental-features "["scale-monitor-framebuffer"]"`
- `gsettings reset org.gnome.mutter experimental-features`

### 4.1.12 Mouse and trackpad handling

FLTK receives information about mouse and pointer events via a ‘listener’ made up of 5 pointers to functions which Wayland calls when events listed in table below occur. These functions receive from Wayland enough information in their parameters to generate corresponding FLTK events, that is, calls to `Fl::handle(int event_type, Fl_Window *)`.

<table>
<thead>
<tr>
<th>listener function</th>
<th>called by Wayland when</th>
<th>resulting FLTK events</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointer_enter</td>
<td>pointer enters a window</td>
<td>FL_ENTER</td>
</tr>
<tr>
<td>pointer_leave</td>
<td>pointer leaves a window</td>
<td>FL_LEAVE</td>
</tr>
<tr>
<td>pointer_motion</td>
<td>pointer moves inside a window</td>
<td>FL_MOVE</td>
</tr>
<tr>
<td>pointer_button</td>
<td>state of mouse buttons changes</td>
<td>FL_PUSH, FL_RELEASE</td>
</tr>
<tr>
<td>pointer_axis</td>
<td>trackpad is moved vertically or horizontally</td>
<td>FL_MOUSEWHEEL</td>
</tr>
</tbody>
</table>

**pointer_listener** is installed by a call to function *wl_pointer_add_listener()* made by function `seat_capabilities()` which is itself another ‘listener’ made up of 2 function pointers

```c
static struct wl_seat_listener seat_listener = {
  seat_capabilities,
  seat_name
};
```

installed by a call to function *wl_seat_add_listener()* made by function `registry_handle_global()` when it receives a "wl_seat" interface.

### Handling middle mouse button clicks on window titlebars

The gnome desktop, via its *gnome-tweaks* application, allows to determine what happens when a middle mouse button click occurs on a window titlebar. To obey this setting, FLTK implements part of the GTK Shell protocol as follows. Mutter, gnome’s Wayland compositor, declares its support of the GTK Shell protocol calling `registry_handle_global()` with its interface argument equal to "gtk_shell1". FLTK initializes then a static global variable `gtk_shell` of type `struct gtk_shell1 *`. Member functions of **pointer_listener** mentioned above run for all mouse events on all *wl_surface* objects. The table above describes what these functions do for mouse events on FLTK-created *wl_surface* objects. But they also run for the libdecor-created *wl_surface* objects corresponding to window titlebars. Thus, member function `pointer_enter()` runs when the mouse enters a titlebar. It calls `Fl_Wayland::Screen_Driver::event_coords_from_surface()` which calls `Fl_Wayland_Window_Driver::surface_to_window()` which, as mentioned above, can distinguish FLTK-created from non FLTK-created *wl_surface* objects. This allows `pointer_enter()` to identify the entered surface as a titlebar and to assign static global variable `gtk_shell_surface` with the titlebar’s *wl_surface* when the mouse enters a titlebar.

Similarly, member function `pointer_leave()` sets `gtk_shell_surface` to NULL when the mouse leaves this titlebar. When there’s a click on a titlebar, member function `pointer_button()` runs this code

```c
if (gtk_shell_surface && state == WL_POINTER_BUTTON_STATE_PRESSED && button == BTN_MIDDLE) {
  struct gtk_surface1 *gtk_surface = gtk_shell1_get_gtk_surface(gtk_shell, gtk_shell_surface);
  gtk_surface1_titlebar_gesture(gtk_surface, serial, seat->wl_seat, GTK_SURFACE1_GESTURE_MIDDLE_CLICK);
  gtk_surface1_release(gtk_surface);
  return;
}
```

which ensures that what *gnome-tweaks* has assigned to middle-click events is executed. At this point, FLTK obeys what libdecor decides for right-click (display the window menu) and double-click (maximize the window) events on titlebars which may diverge from *gnome-tweaks* settings.
4.1.13 Wayland cursors

Wayland defines types struct \texttt{wl\_cursor} and struct \texttt{wl\_cursor\_theme} to hold cursor-related data. FLTK uses function \texttt{init\_cursors()} from file \texttt{Fl\_Wayland\_Screen\_Driver.cxx} to obtain the 'cursor theme' name using function \texttt{libdecor\_get\_cursor\_settings()} of library libdecor. Function \texttt{wl\_cursor\_theme\_load()} then returns a pointer to an object of type struct \texttt{wl\_cursor\_theme} stored in member variable \texttt{cursor\_theme} of the \texttt{Fl\_Wayland\_Screen\_Driver\::seat} record. Function \texttt{init\_cursors()} is itself called by a 'listener' called \texttt{seat\_capabilities()} installed when function \texttt{registry\_handle\_global()} receives a "wl\_seat" interface, at program startup. It is also called when the value of the Wayland scaling factor changes: \texttt{output\_done()} calls \texttt{try\_update\_cursor()} calls \texttt{init\_cursors()}. Function \texttt{output\_done()} belongs to a 'listener' installed when function \texttt{registry\_handle\_global()} receives a "wl\_output" interface.

Each time \texttt{Fl\_Window\::cursor(Fl\_Cursor)} runs, FLTK calls \texttt{Fl\_Wayland\_Window\_Driver\::set\_cursor(Fl\_Cursor)} which calls \texttt{wl\_cursor\_theme\_get\_cursor()} to set the current cursor shape to one of the standard shapes from the \texttt{Fl\_Cursor} enumeration. This Wayland function selects a cursor shape based on the current \texttt{wl\_cursor\_theme} object and a cursor name and returns a pointer to a struct \texttt{wl\_cursor}. Under the gnome desktop, cursor names are the files of directory /usr/share/icons/\texttt{XXXX}/cursors/ where \texttt{XXXX} is the 'gnome cursor theme' (default= Adwaita). For example, what FLTK calls \texttt{FL\_CURSOR\_INSERT} corresponds to file xterm therein. The full correspondence between \texttt{Fl\_Cursor} values and names of files therein is found in function \texttt{Fl\_Wayland\_Window\_Driver\::set\_cursor(Fl\_Cursor)}.

FLTK stores in member variable \texttt{default\_cursor} of the \texttt{Fl\_Wayland\_Screen\_Driver\::seat} record a pointer to the currently used \texttt{wl\_cursor} object, and the current \texttt{Fl\_Cursor} value in member \texttt{standard\_cursor} of the \texttt{Fl\_Wayland\_Window\_Driver} object.

Finally, function \texttt{do\_set\_cursor()} of file \texttt{Fl\_Wayland\_Screen\_Driver.cxx} makes the system pointer use the current \texttt{wl\_cursor} object to draw its shape on screen. That's done with a call to \texttt{wl\_pointer\_set\_cursor()} and a few other functions.

Custom cursor shapes

To support custom cursors, FLTK presently uses a non-public type, struct \texttt{cursor\_image}, defined in file \texttt{Fl\_Wayland\_Window\_Driver.cxx} as follows:

\begin{verbatim}
struct cursor_image {
  struct wl\_cursor\_image\_image;
  struct wl\_cursor\_theme\_theme;
  struct wl\_buffer\ 
    buffer;
  int offset;
};
\end{verbatim}

This definition has been copied to the FLTK source code from file \texttt{wayland\_cursor\_c} of the Wayland project source code because it's not accessible via Wayland header files. It shows that a pointer to a \texttt{cursor\_image} object can also be viewed as a pointer to the embedded struct \texttt{wl\_cursor\_image\_object}, this one being part of the public Wayland API. It also shows that a struct \texttt{cursor\_image\_object} has an associated struct \texttt{wl\_buffer\_object} used to contain the cursor's graphics.

Function \texttt{Fl\_Wayland\_Window\_Driver\::set\_cursor(const Fl\_RGB\_Image \*rgb, int hotx, int hoty)} gives FLTK support of custom cursor shapes. It calls \texttt{Fl\_Wayland\_Window\_Driver\::set\_cursor\_4args()} that creates a \texttt{cursor\_image\_object}, allocates the corresponding \texttt{wl\_buffer\_object} by a call to \texttt{Fl\_Wayland\_Graphics\_Driver\::create\_shm\_buffer()} via \texttt{create\_wl\_buffer()} and \texttt{create\_wl\_buffer()} and draws the cursor shape into that buffer using the offscreen-drawing method of FLTK.

The public type struct \texttt{wl\_cursor} is essentially an array of \texttt{wl\_cursor\_image\_object}s and a name:

\begin{verbatim}
struct wl\_cursor {
  unsigned int image\_count;
  struct wl\_cursor\_image \*images;
  char \*name;
};
\end{verbatim}

Function \texttt{Fl\_Wayland\_Window\_Driver\::set\_cursor\_4args()} also creates a struct \texttt{wl\_cursor} object containing a single \texttt{wl\_cursor\_image\_object}, which is in fact the \texttt{cursor\_image}. Finally, a struct \texttt{custom\_cursor\_object} (see \texttt{wl\_window}) is allocated and used to memorize the struct \texttt{wl\_cursor} and the cursor's image and hotspot. A pointer to this struct \texttt{custom\_cursor\_object} is stored in member \texttt{custom\_cursor\_object} of the window's \texttt{wl\_window}.

Function \texttt{Fl\_Wayland\_Window\_Driver\::set\_cursor\_4args()} is also called when a window with a custom cursor is moved between distinct displays or when a display is rescaled to adapt the cursor size to the new display's scale factor.

Member function \texttt{Fl\_Wayland\_Window\_Driver\::delete\_cursor\_object()} is used to delete any custom curs-
sor shape. This occurs when a window associated to a custom cursor is un-mapped and when such a window gets associated to a standard cursor or to a new custom cursor.

### 4.1.14 Keyboard support

The "Mouse handling" section above mentioned function `seat_capabilities()` that Wayland calls when the app discovers its "seat". Presence of flag `WL_SEAT_CAPABILITY_KEYBOARD` in argument `capabilities` of this function indicates that a keyboard is available. In that case, a call to `wl_seat_get_keyboard()` returns a pointer stored in member `wl_keyboard` of the `Fl_Wayland_Screen_Driver::seat` object, and a call to `wl_keyboard_add_listener()` installs a 6-member listener of type `struct wl_keyboard_listener`.

These 6 FLTK-defined, callback functions are used as follows.

Function `wl_keyboard_keymap()` runs when the app starts and also if the keyboard layout is changed during run-time. It allows initialization of access to this keyboard. Noticeably, member `xkb_state` of type `struct xkb_state` of the current `Fl_Wayland_Screen_Driver::seat` record is adequately initialized.

Functions `wl_keyboard_enter()` and `wl_keyboard_leave()`, called when focus enters and leaves a surface, send `FL_FOCUS` and `FL_UNFOCUS` events to the `Fl_Window` object corresponding to this surface.

Function `wl_keyboard_key()` runs each time a keyboard key is pressed or released. Its argument `key`, to which 8 must be added, provides the keycode via function `xkb_state_key_get_one_sym()` and then the corresponding text via function `xkb_state_key_get_utf8()` which is put in `Fl::e_text`. Then, a few calls to functions whose name begin with `xkb_` are necessary to support dead and compose keys.

Finally a call to `Fl::handle()` sends an `FL_KEYDOWN` or `FL_KEYUP` event to the appropriate `Fl_Window`.

Also, function `wl_keyboard_modifiers()` runs when a modifier key (e.g., shift, control) is pressed or released. Calls to functions `xkb_state_update_mask()` and `xkb_state_mod_name_is_active()` allow FLTK to set `Fl::e_state` adequately.

Function `wl_keyboard_repeat_info()` does not run, for now, because this would require version 4 of the `wl_keyboard` object which is at version 2 in all tested Wayland compositors.

### 4.1.15 Support of text input methods

When the connected Wayland compositor supports text input methods, function `registry_handle_global()` gets called with its interface argument equal to `zwp_text_input_manager_v3_global()`.

The following call to `wl_registry_bind()` returns a pointer to type `struct zwp_text_input_manager_v3` that is stored as member `text_input_base` of the `Fl_Wayland_Screen_Driver::seat` object.

Later, when function `seat_capabilities()` runs, `text_input_base` is found not NULL, which triggers a call to function `zwp_text_input_manager_v3_get_text_input()` returning a value of type `struct text_input_base` * and stored as member `text_input` of the `Fl_Wayland_Screen_Driver::seat` object.

Next, a call to `zwp_text_input_v3_add_listener()` associates this `text_input` with a 6-member listener of type `struct text_input_v3_listener`. These 6 FLTK-defined, callback functions are used as follows.

Functions `text_input_preedit_string()` and `text_input_commit_string()` are called when the text input method asks the client app to insert 'marked' text or regular text, respectively. Complex text input often begins by inserting temporary text which is said to be 'marked' before replacing it with the text that will stay in the document. FLTK underlines marked text to distinguish it from regular text.

Functions `text_input_delete_surrounding_text()` and `text_input_done()` have no effect at present, without this preventing input methods that have been tested with FLTK from working satisfactorily.

It's necessary to inform text input methods of the current location of the insertion point in the active surface. This information allows them to map their auxiliary windows next to the insertion point, where they are expected to appear. The flow of information on this topic is as follows:

- The two FLTK widgets supporting text input, `Fl_Input_` and `Fl_Text_Display`, transmit to FLTK the window coordinates of the bottom of the current insertion point and the line height each time they change calling function `Fl_set_spot()`.
• `fl_set_spot()` calls the platform override of virtual member function `Fl_Screen_Driver::set_spot()`. Under Wayland, this just calls `Fl_Wayland_Screen_Driver::insertion_point_location(int x, int y, int height)` which calls `zwp_text_input_v3_set_cursor_rectangle()` to inform the text input method about the surface position and size of the insertion point and also memorizes this information in static member variables of class `Fl_Wayland_Screen_Driver`.

• Callback function `text_input_enter()` calls `Fl_Wayland_Screen_Driver::insertion_point_location(int *x, int *y, int *height)` which gives it the stored position information, and then calls `zwp_text_input_v3_set_cursor_rectangle()` to inform the text input method about the position of the insertion point.

### 4.1.16 Interface with libdecor

FLTK uses a library called *libdecor* to determine whether the Wayland compositor uses CSD or SSD mode, and also to draw window titlebars when in CSD mode (see *libdecor*). *Libdecor* is conceived to be present in a shared library linked to the Wayland client application which itself, and if the running Wayland compositor uses CSD mode, loads another shared library intended to draw titlebars in a way that best matches the Desktop. As of late 2023, libdecor is at version 0.2.0 and contains two titlebar-drawing plugins:

- `libdecor-gtk` intended for the Gnome desktop;
- `libdecor-cairo` for other situations.

Because `libdecor` is not yet in major Linux packages, or only at version 0.1.x, FLTK bundles the most recent source code of `libdecor` and its plugins. This code is included in libfltk. FLTK uses `libdecor-gtk` when software package `libgtk-3-dev` is present in the build system, and `libdecor-cairo` otherwise.

As of late 2023, libdecor version 0.2.0 is available in very recent Linux distributions in packages `libdecor-0-dev` and `libdecor-0-plugin-1-gtk`. If they are installed on the build system, preprocessor variable `USE_SYSTEM_LIBDECOR` 1 is defined and points to the name of a directory, this directory is searched for a potential `libdecor` plugin in the form of a shared library; if one is found, FLTK and their plugins are loaded at run-time from shared libraries.

Whatever the value of `OPTION_USE_SYSTEM_LIBDECOR`, FLTK and libdecor use environment variable `LIBDECOR_PLUGIN_DIR` as follows: if this variable is defined and points to the name of a directory, this directory is searched for a potential `libdecor` plugin in the form of a shared library; if one is found, FLTK and `libdecor` load it and use it.

The `libdecor` source code bundled in FLTK is identical to that of the `libdecor` repository. Nevertheless, FLTK uses this code with some minor changes. For example, except if `USE_SYSTEM_LIBDECOR` is 1, FLTK needs to modify function `libdecor_new()` charged of loading the plugin, to make it use the plugin code that is included in libfltk if none is found as a dynamic library. This is done as follows in file `libdecor/build/fl_libdecor.c`:

```c
#include "libdecor_new.orig"
#include ".//src/libdecor.c"
#undef libdecor_new

void libdecor_new() { // FLTK rewrite of this function
  .......
}
```

FLTK compiles file `fl_libdecor.c` which includes `libdecor.c` to the effect that all of the `libdecor` code becomes part of libfltk except that function `libdecor_new()` is substituted by its FLTK rewrite, without file `libdecor.c` being modified at all. This trick is also used to modify function `libdecor_frame_set_minimized()` to bypass a bug in the Weston compositor before version 10. Similarly, FLTK compiles file `fl_libdecor-plugins.c` which includes either `libdecor-gtk.c` or `libdecor-cairo.c` to the effect that the desired plugin becomes part of libfltk.

To support function `Fl_Widget_Surface::draw_decorated_window()` that draws a mapped window and its titlebar, FLTK needs to perform two operations: 1) identify what plugin is operating, and 2) call a function that is specific of that plugin and that returns the pixels of the drawn titlebar.

FLTK performs operation 1) above using its function `get_libdecor_plugin_description()` of file `fl_libdecor-plugins.c` that returns a human readable string describing the running plugin. Each plugin puts
its own string in member description of a record of type struct libdecor_plugin_description. Although this type is public in header file libdecor-plugin.h, accessing the symbol defined by the plugin to store a pointer to a value of this type is complicated for a reason and solved by a method detailed in a comment before function get_libdecor_plugin_description().

Operation 2) above is done by FLTK-defined function fl_libdecor_titlebar_buffer() from file fl-_libdecor-plugins.c. This function calls get_libdecor_plugin_description() seen above to get the running plugin’s descriptive string. That is "GTK3 plugin" with libdecor-gtk. FLTK function gtk_titlebar_buffer() is then called, and returns a pointer to the start of a byte buffer containing the titlebar graphics. That is, again, not possible with the public libdecor API. Therefore, FLTK copies to fl_libdecor-plugins.c the definitions of several types given in libdecor-gtk.c or libdecor-cairo.c such as type struct border_component.

4.1.17 Copy/Paste/Drag-n-Drop

FLTK follows the procedure that is very well described in item "Wayland clipboard and drag & drop" of the Documentation resources. All corresponding source code is in file src/drivers/Wayland/fl_wayland-_clipboard_dnd.cxx.

This part of the Fl_Wayland_Screen_Driver::seat record stores pointers to Wayland objects used for clipboard and D-n-D operations:

struct wl_data_device_manager *data_device_manager;
struct wl_data_device *data_device;
struct wl_data_source *data_source;

FLTK can copy or paste plain UTF-8 text or image data to/from the clipboard. Images are copied to the clipboard as image/bmp mime type. Images in image/bmp or image/png mime types from the clipboard can be pasted to FLTK apps.

Files dropped are received one pathname per line, with no ‘\n’ after the last pathname.

4.1.18 EGL as support for OpenGL

Wayland uses EGL™ to interface OpenGL with the underlying native platform window system. OpenGL-using FLTK apps are therefore linked to libwayland-egl.so and libEGL.so in addition to libGL.so and libGLU.so.

EGL is initialized calling member function Fl_Wayland_GL_Window_Driver::init() once, the first time the Fl_Wayland_GL_Window_Driver c’tor runs. That is done with calls to eglGetDisplay(), eglInit(), and eglBindAPI().

Member function Fl_Wayland_GL_Window_Driver::find() calls eglChooseConfig() to filter the set of GL configurations that match the Fl_Gl_Window’s mode(), and puts in the returned Fl_Gl_Object the first matching configuration. The filtering gets done with bits EGL_WINDOW_BIT, to support the creation of window surfaces, and EGL_OPENGL_BIT, to support the creation of OpenGL contexts.

EGL needs 2 more objects created for each Fl_Gl_Window. They have types struct wl_eegl_window and EGSurface, and are created by member function Fl_Wayland_GL_Window_Driver::make_eegl_window() which runs at the beginning of Fl_Gl_Window::make_current(). The first argument of the call to wl_eegl_window_create() therein has type struct wl_surface * and is what connects EGL with the targeted Wayland window.

EGL creates with eglCreateContext() an object of type EGLContext via member function Fl_Wayland_GL_Window_Driver::create gl_context() called by Fl_Gl_Window::make_current(). Types EGLContext and GLContext are 2 names for the same object. The call to eglCreateContext() is made asking for a GL context of version at least 2. This does not prevent from obtaining contexts of higher versions, namely above 3.2, which are compatible with version 2 (the so-called compatibility profile) under all tested Linux systems.

FLTK function Fl_Gl_Window::make_current() calls overridden function Fl_Wayland_Gl_Window_Driver::set_gl_context() which calls EGL function eglMakeCurrent() when the cached context changes.

FLTK calls function Fl_Wayland_GL_Window_Driver::swap_buffers() each time it wants a GL context to be sent to the display. This function contains some pure GL code to emulate an overlay buffer to support Fl_Gl_Window objects overriding their draw_overlay() member function. Then, it calls function eglSwapBuffers().

The overridden Fl_Wayland_GL_Window_Driver::resize() function is implemented with calls to wl_eegl_window_get_attached_size() and wl_eegl_window_resize().

Generated by Doxygen
4.1 The Wayland backend for its developer

Class Fl_Wayland_GL_Plugin exists to allow libfltk to call functions from libfltk_gl, libwayland-egl.so or libEGL.so and without having libfltk force linking any FLTK app with these GL-related libraries. For example, Fl_Wayland_Window_Driver::flush() needs to call Fl_Gl_Window::valid(0).

4.1.19 FLTK-defined, Wayland-specific types

struct wld_window

Defined in Fl_Wayland_Window_Driver.H. One such record is created for each shown()’n Fl_Window by Fl_Wayland_Window_Driver::makeWindow(). Function fl_wl_xid(Fl_Window *) returns a pointer to the struct wld_window of its argument.

struct wld_window {
    Fl_Window *fl_win;
    struct wl_list outputs; // linked list of displays where part or whole of window maps
    struct wl_surface *wl_surface; // the window’s surface
    struct wl_callback *frame_cb; // non-NULL until Wayland can process new surface commit
    struct Fl_Wayland_Graphics_Driver::wld_buffer *buffer; // see wld_buffer
    struct xdg_surface *xdg_surface; // the window’s surface
    enum Fl_Wayland_Window_Driver::kind kind; // DECORATED or POPUP or SUBWINDOW or UNFRAMED
    union {
        struct libdecor_frame *frame; // for DECORATED windows
        struct wl_subsurface *subsurface; // for SUBWINDOW windows
        struct xdg_popup *xdg_popup; // for POPUP windows
        struct xdg_toplevel *xdg_toplevel; // for UNFRAMED windows
    };
    struct custom_cursor_ {
        struct wl_cursor *wl_cursor;
        const Fl_RGB_Image *rgb;
        int hotx, hoty;
    } *custom_cursor; // non-null when using custom cursor
    int configured_width; // used when negotiating window size with the compositor
    int configured_height;
    int floating_width; // helps restoring size after un-maximizing
    int floating_height;
    int state; // indicates whether window is fullscreen, maximized. Used otherwise for POPUPs
    bool covered; // specially for Mutter and issue #878
};

struct Fl_Wayland_Graphics_Driver::draw_buffer

Defined in file Fl_Wayland_Graphics_Driver.H. One such record is created when an Fl_Image_Surface object is created. One such record is also embedded inside each struct Fl_Wayland_Graphics_Driver::wld_buffer record (see wld_buffer).

struct Fl_Wayland_Graphics_Driver::draw_buffer {
    unsigned char *buffer; // address of the beginning of the Cairo image surface’s byte array
    cairo_t *cairo_; // used when drawing to the Cairo image surface
    size_t data_size; // of buffer and wl_buffer, in bytes
    int stride; // bytes per line
    int width; // in pixels
};

FLTK gives offscreen buffers the platform-dependent type Fl_Offscreen which is in fact member cairo_ of struct Fl_Wayland_Graphics_Driver::draw_buffer. Thus, a variable with type Fl_Offscreen needs be cast to type cairo_t*. Static member function struct draw_buffer *offscreen_← buffer(Fl_Offscreen) of class Fl_Wayland_Graphics_Driver returns the draw_buffer record corresponding to an Fl_Offscreen value.

struct Fl_Wayland_Graphics_Driver::wld_buffer

Defined in file Fl_Wayland_Graphics_Driver.H. One such record is created by Fl_Wayland_Graphics_Driver::create_wld_buffer() when an Fl_Window is show()’n or resized, when a custom
cursor shape is created, or when text is dragged.

```c
struct Fl_Wayland_Graphics_Driver::wld_buffer {
    struct draw_buffer draw_buffer; // see draw_buffer
    struct wl_list link; // links all buffers from the same wl_shm_pool
    struct wl_buffer *wl_buffer; // the Wayland buffer
    void *data; // address of the beginning of the Wayland buffer's byte array
    struct wl_shm_pool *shm_pool; // pter to wl_shm_pool from which this wl_buffer comes
    bool draw_buffer_needs_commit; // true when draw_buffer has been modified but not yet committed
    bool in_use; // true while being committed
    bool released; // true after buffer_release() was called
};
```

```c
struct Fl_Wayland_Screen_Driver::output

Defined in Fl_Wayland_Screen_Driver.H. One such record is created for each display of the system by function registry_handle_global() when it receives a "wl_output" interface. These records are kept in a linked list of them all, and an identifier of this linked list is stored in member outputs of the unique Fl_Wayland_Screen_Driver object FLTK uses. Thus, Fl_Wayland_Screen_Driver *scr_driver = (Fl_Wayland_Screen_Driver*)Fl::screen_driver();
struct wl_list list_of_all_displays = scr_driver->outputs;
gives access, the Wayland way, to the linked list of displays in the system.

```c
struct Fl_Wayland_Screen_Driver::output { // one record for each display
    uint32_t id; // an identifier of the display
    int x, y; // logical position of the top-left of display
    int width; // nber of horizontal pixels
    int height; // nber of vertical pixels
    float dpi; // at this point, always 96.
    struct wl_output *wl_output; // the Wayland object for this display
    int wld_scale; // Wayland scale factor
    float gui_scale; // FLTK scale factor
    bool done; // true means record members have been initialized
    struct wl_list link; // links these records together
};
```

It's possible to get the FLTK-defined record associated to a display from the Wayland-associated object for the same display, say struct wl_output *wl_output, by this call: (struct Fl_Wayland_Screen_Driver::output *)wl_output_get_user_data(wl_output).

```c
struct Fl_Wayland_Screen_Driver::seat

Defined in file Fl_Wayland_Screen_Driver.H. One record is created by function registry_handle_global() when it receives a "wl_seat" or wl_data_device_manager_interface.name interface. A pointer to this struct is stored in member seat of the client's unique Fl_Wayland_Screen_Driver object.

```c
struct Fl_Wayland_Screen_Driver::seat {
    struct wl_seat *wl_seat;
    struct wl_pointer *wl_pointer;
    struct wl_keyboard *wl_keyboard;
    uint32_t keyboard_enter_serial;
    struct wl_surface *keyboard_surface;
    struct wl_list pointer_outputs;
    struct wl_cursor_theme *cursor_theme;
    struct wl_cursor *default_cursor;
    struct wl_surface *cursor_surface;
    struct wl_surface *pointer_focus;
    int pointer_scale;
    uint32_t serial;
    uint32_t pointer_enter_serial;
    struct wl_data_device_manager *data_device_manager;
    struct wl_data_device *data_device;
    struct wl_data_source *data_source;
    struct xkb_state *xkb_state;
};
```
struct xkb_context *xkb_context;
struct xkb_keymap *xkb_keymap;
struct xkb_compose_state *xkb_compose_state;
char *name;
struct zwp_text_input_v3 *text_input;
};

4.1.20 Documentation resources

| The Wayland book | Extensive introduction to Wayland programming written by the author of the sway compositor, unfortunately unachieved. |
| Wayland Explorer | Documentation of all Wayland protocols, both stable and unstable. A language-independent syntax is used which makes function names usable from C or C++ not always obvious. Some useful functions seem undocumented here for an unclear reason. |
| Wayland clipboard and drag & drop | Detailed explanation of how clipboard and drag-and-drop work under Wayland. |
| Wayland and input methods | Blog article introducing to the issue of text input methods under Wayland. |
| Input Method Hub | Entry page for input method support giving newcomers a first understanding of what input methods are and how they are implemented in Wayland. |

4.2 Developer info for bundled libs

This chapter details the procedure to update the libraries which are bundled inside FLTK.

4.2.1 Introduction

This file is mainly intended for FLTK developers and contains information about the current versions of all bundled libraries and about how to upgrade these bundled libraries.

Starting with FLTK 1.4.0 the bundled libraries jpeg, png, and zlib use "symbol prefixing" with the prefix 'fltk_' for all external symbols to distinguish the bundled libraries from existing system libraries and to avoid runtime errors.

User code compiled correctly with the header files provided by the bundled image libraries need not be changed. The nanosvg library is not affected.

4.2.2 Current status

Current versions of bundled libraries (as of December 5, 2023):

<table>
<thead>
<tr>
<th>Library</th>
<th>Version/git commit</th>
<th>Release date</th>
<th>FLTK Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>jpeg</td>
<td>jpeg-9e</td>
<td>2022-01-16</td>
<td>1.4.0</td>
</tr>
<tr>
<td>nanosvg</td>
<td>7aeda550a8 [1]</td>
<td>2023-12-02</td>
<td>1.4.0</td>
</tr>
<tr>
<td>png</td>
<td>libpng-1.6.40</td>
<td>2023-06-21</td>
<td>1.4.0</td>
</tr>
<tr>
<td>zlib</td>
<td>zlib-1.3</td>
<td>2023-08-18</td>
<td>1.4.0</td>
</tr>
<tr>
<td>libdecor</td>
<td>060fe761 [2]</td>
<td>2023-12-01</td>
<td>1.4.0</td>
</tr>
</tbody>
</table>

Previous versions of bundled libraries (FLTK 1.3.x):

<table>
<thead>
<tr>
<th>Library</th>
<th>Version</th>
<th>Release date</th>
<th>FLTK Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>jpeg</td>
<td>jpeg-9e</td>
<td>2022-01-16</td>
<td>1.3.9</td>
</tr>
<tr>
<td>png</td>
<td>libpng-1.6.40</td>
<td>2023-06-21</td>
<td>1.3.9</td>
</tr>
<tr>
<td>zlib</td>
<td>zlib-1.3</td>
<td>2023-08-18</td>
<td>1.3.9</td>
</tr>
<tr>
<td>jpeg</td>
<td>jpeg-9d</td>
<td>2020-01-12</td>
<td>1.3.6 - 1.3.8</td>
</tr>
<tr>
<td>png</td>
<td>libpng-1.6.37</td>
<td>2019-04-14</td>
<td>1.3.6 - 1.3.8</td>
</tr>
<tr>
<td>zlib</td>
<td>zlib-1.2.11</td>
<td>2017-01-15</td>
<td>1.3.6 - 1.3.8</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Development of the FLTK library

---

   See also git tag 'fltk_yyyy-mm-dd' where yyyy-mm-dd == "Release date"
   and file nanosvg/README.txt.

General information:

FLTK does not include the entire library distributions. We only provide the source files necessary to build the FLTK library and some README and/or CHANGELOG files. There are no test programs or other contributed files. We use our own build files, hence a few files MUST NOT be upgraded when the library source files are upgraded. We strive to keep changes to the library source files as small as possible. Patching library code to work with FLTK should be a rare exception. Symbol prefixing with prefix 'fltk_' is one such exception to the rule.

If patches are necessary all changes in the library files should be marked with "FLTK" in a comment so a developer who upgrades the library later is aware of changes in the source code for FLTK. Look for 'FLTK' and/or 'fltk_' to find the differences.

Additional comments should be added to show the rationale, i.e. why a particular change was necessary. If applicable, add a reference to a Software Trouble Report, GitHub Issue or Pull Request (PR) like “STR 3456”, “Issue #123”, or "PR #234”.

4.2.3 How to update the bundled libraries

It is generally advisable to use a graphical merge program. I’m using ‘meld’ under Linux, but YMMV.

Do not add any source files unless they are required to build the library.

Some config header files may be pre-generated in the FLTK sources. These header files should be left untouched, but it may be necessary to update these files if new items were added to the new library version. In this case the new header should be pre-generated on a Linux system with default options unless otherwise mentioned below for a specific library. Currently there are no known exceptions.

Merging source files:

Please check if some source and header files contain "FLTK" comments and/or 'fltk_' symbol prefixing to be aware of necessary merges. It is also good to download the distribution tar ball or Git source files of the previous version and to run a (graphical) diff or merge tool on the previous version and the bundled version of FLTK to see the “previous” differences.

Files that were not patched in previous versions should be copied to the new version w/o changes. Files that had FLTK specific patches must be merged manually. FLTK patches should be verified (if still necessary) and should be kept in the new source files.

Source and header files that have been added in the new library version should be added in FLTK as well if they are necessary to build the library. A simple "trial and error" should be sufficient to find files that need to be added. Added files must be added to FLTK’s build files as well, usually to both ‘Makefile’ and ‘CMakeLists.txt’ to be used in configure/make and in CMake based builds, respectively.

Upgrade order:

There is only one dependency between all bundled libraries: libpng depends on zlib. Hence zlib should be upgraded first, then all other libs can be upgraded in arbitrary order.

Tests after merge:

Tests should be done on as many platforms as possible, both with autotools (configure/make) and CMake. Windows (Visual Studio) and macOS (Xcode) builds need CMake to generate the IDE files.

Upgrade notes for specific libraries:

The following chapters contain informations about specific files and how they are upgraded. Since the changes in all bundled libraries are not known in advance this information may change in the future. Please verify that no other changes are necessary.

---

Generated by Doxygen
4.2.4 zlib:

Website: [https://zlib.net/](https://zlib.net/)
Download: See website and follow links.
Repository: git clone [https://github.com/madler/zlib.git](https://github.com/madler/zlib.git)

Zlib should be upgraded first because libpng depends on zlib.

Download the latest zlib sources, `cd` to `/path/to/zlib` and run

```
$ ./configure --zprefix
```

This creates the header file `zconf.h` with definitions to enable the standard `z_` symbol prefix.

Unfortunately zlib requires patching some source and header files to convert this `z_` prefix to `fltk_z_` to be more specific. As of this writing (Nov. 2021) three files need symbol prefix patches:

- `gzread.c`
- `zconf.h`
- `zlib.h`

You may want to compare these files and/or the previous version to find out which changes are required. The general rule is to change all occurrences of `z_` to `fltk_z_` but there are exceptions.

The following files need special handling:

- `CMakeLists.txt`: Keep FLTK version, update manually if necessary.
- `Makefile`: Same as `CMakeLists.txt`
- `gzread.c`: Merge changes (see above, manual merge recommended).
- `zconf.h`: Merge changes (see above, manual merge recommended).
- `zlib.h`: Merge changes (see above, manual merge recommended).
- `makedepend`: Keep this file.

Run `make depend` in the zlib folder on a Linux system after the upgrade to update this file.

4.2.5 png:

Website: [http://libpng.org/pub/png/libpng.html](http://libpng.org/pub/png/libpng.html)
Download: See website and follow links.
Repository: git clone [https://git.code.sf.net/p/libpng/code libpng](https://git.code.sf.net/p/libpng/code)

Libpng should be upgraded after zlib because it depends on zlib.

Download the latest libpng sources, `cd` to `/path/to/libpng` and run

```
$ ./configure --with-libpng-prefix=fltk_
$ make
```

This creates the header files `pnglibconf.h` and `pngprefix.h` with the `fltk_` symbol prefix.

The following files need special handling:

- `CMakeLists.txt`: Keep FLTK version, update manually if necessary.
- `Makefile`: Same as `CMakeLists.txt`
- `pnglibconf.h`: Generate on a Linux system and merge (see above).
- `pngprefix.h`: Generate on a Linux system and merge (see above).
- `makedepend`: Keep this file.
- `png.c`: Keep a change labelled with "FLTK"
- `pngerror.c`: Keep two changes labelled with "FLTK"

Run `make depend` in the png folder on a Linux system after the upgrade to update this file.
4.2.6 jpeg:

Website:  https://ijg.org/
Download: See website and follow links.
Repository: N/A

Download the latest jpeg-xy sources on a Linux (or Unix) system, `cd` to /path-to/jpeg-xy and run
$ ./configure
$ make [-jN]

This builds the library and should create the static library file `.libs/libjpeg.a'.

Execute the following command to extract the libjpeg symbol names used to build the 'prefixed' libfltk_jpeg library:
$ nm --extern-only --defined-only .libs/libjpeg.a | awk '{print $3}' \
| sed '/^$/d' | sort -u | awk '{print "#define "$1 "fltk_"$1}'}
> fltk_jpeg_prefix.h

This creates the header file 'fltk_jpeg_prefix.h' with the '# define' statements using the 'fltk_' symbol prefix.

The following files need special handling:

- CMakeLists.txt: Keep FLTK version, update manually if necessary.
- Makefile: Same as CMakeLists.txt.
- fltk_jpeg_prefix.h: Generate on a Linux system and merge (see above).
- jconfig.h: keep changes flagged with
  /* FLTK */
  Note: more to come...
- makedepend: Keep this file.

Run 'make depend' in the jpeg folder on a Linux system after the upgrade to update this file.

4.2.7 nanosvg:

Website:  https://github.com/memononen/nanosvg
Download: See website and follow links.
Repository: git clone https://github.com/memononen/nanosvg.git
FLTK Fork: git clone https://github.com/fltk/nanosvg.git

FLTK has its own GitHub fork of the original repository (see above).

The intention is to update this fork from time to time so the FLTK specific patches are up-to-date with the original library. Hopefully the FLTK patches will be accepted upstream at some time in the future so we no longer need our own patches. AlbrechtS, 04 Feb 2018.

Update (Feb 22, 2021): The upstream library is officially no longer maintained (see README.md) although updates appear from time to time.

Use this fork (branch 'fltk') to get the nanosvg library with FLTK specific patches:
$ git clone https://github.com/fltk/nanosvg.git nanosvg-fltk
$ cd nanosvg-fltk
$ git checkout fltk
$ cd src
$ cp nanosvg.h nanosvgrast.h /path/to/fltk-1.4/nanosvg/

This library does not have its own build files since it is a header-only library. The headers are included in FLTK where necessary.

The following files need special handling:
- nanosvg.h: Merge or download from FLTK's fork (see above).
- nanosvgrast.h: Merge or download from FLTK's fork (see above).

Maintaining branch 'fltk' in FLTK's fork of nanosvg (fltk/nanosvg):

Only maintainers with write access on fltk/nanosvg can do this. Others can fork our fltk/nanosvg fork in their own GitHub account and either open a PR on fltk/nanosvg or tell us about their changes in fltk.development.

Use something similar to the following commands to update FLTK's fork of nanosvg to the latest version. Commands are only examples, you may need to change more or less, depending on the outstanding updates.

Step 1: clone the fltk/nanosvg fork, set the remote 'upstream', and update the 'master' branch:
$ cd /to/your/dev/dir
$ git clone https://github.com/fltk/nanosvg.git nanosvg-fltk
$ cd nanosvg-fltk
$ git remote add upstream https://github.com/memononen/nanosvg
$ git checkout master
$ git pull upstream master
Note: the 'master' branch must never be changed, i.e. it must always be the same as 'upstream/master'. Never commit your own (FLTK specific) changes to branch 'master'.

Step 2: rebase branch 'fltk' on the new master (upstream/master), fix potential conflicts, and tag the new branch. It is important to keep the individual FLTK specific patches intact (one commit per patch) because this will preserve the history and the committer and make it easier to skip single patches when they are accepted upstream.

$ git checkout fltk
$ git rebase upstream/master

At this point you may need to fix conflicts! Do whatever is necessary to update the branch 'fltk'. Now 'git tag' the 'fltk' branch for later reference. Hint: use 'git show <any-older-tag-name>' to see its contents. I like to write a summary of commits in the tag comment.

$ git tag -a fltk_yyyy-mm-dd fltk

Replace 'yyyy-mm-dd' with the current date and add a comment when asked for it (your editor will open an empty file).

Step 3: at this point it is recommended to copy the changed header files to your working copy of the FLTK library and test the changes. If anything is wrong, go back, fix the bugs and change the git tag (delete and create a new one).

Step 4: push the new branch 'fltk' and the tag to the fltk/nanosvg repository:

$ git push -f origin fltk
$ git push origin fltk_yyyy-mm-dd

Step 5: copy the changed files to your working copy of the FLTK repository (if not done already), update this file accordingly, and commit/push the update to the fltk/fltk repository.

4.2.8 libdecor:

Website: https://gitlab.freedesktop.org/libdecor/libdecor
Download: See website and follow links.
Repository: git clone https://gitlab.freedesktop.org/libdecor/libdecor.git

libdecor is used by the Wayland/X11 hybrid platform to draw window titlebars when FLTK apps run as Wayland clients and the running Wayland compositor uses client-side decoration. In the future, when libdecor will have made its way into Linux packages, FLTK will use the system version of libdecor. libdecor will remain as an FLTK bundle to support Linux configurations where the libdecor package is not available or not installed.

FLTK uses libdecor source files without any modification. This part of the libdecor source tree is copied to directory libdecor/ of the FLTK source tree:

    LICENSE
    README.md
    src/ ... and files below except meson.build files

Furthermore, directory libdecor/build/ of the FLTK source tree does not originate from the libdecor source tree but contains 3 FLTK-created files. File build/Makefile may need changes if a libdecor update adds or renames source files.

4.3 Developer Information

This chapter describes FLTK development and documentation.

Example

```c
/** \file
 * Fl_Clock, Fl_Clock_Output widgets. */

/**
 * Fl_Clock_Output
 * \brief This widget can be used to display a program-supplied time.
 * The time shown on the clock is not updated. To display the current time,
 * use Fl_Clock instead.

 * \image html clock.png
 * \image latex clock.png " width=10cm
 * \image html round_clock.png
 * \image latex clock.png " width=10cm
```
Development of the FLTK library

** Returns the displayed time.
Returns the time in seconds since the UNIX epoch (January 1, 1970).

```cpp
ulong value() const {return value_;
```

** Set the displayed time.
Set the time in seconds since the UNIX epoch (January 1, 1970).

```cpp
void Fl_Clock_Output::value(ulong v) {
    ...
}
```

** Create an Fl_Clock widget using the given position, size, and label string.
The default boxtype is Fl_NO_BOX.

```cpp
Fl_Clock::Fl_Clock(int X, int Y, int W, int H, const char *L) :
    Fl_Clock_Output(X, Y, W, H, L) {
```

** Create an Fl_Clock widget using the given boxtype, position, size, and
label string.

```cpp
Fl_Clock::Fl_Clock(uchar t, int X, int Y, int W, int H, const char *L) :
    Fl_Clock_Output(X, Y, W, H, L) {
    type(t);
    box(t==FL_ROUND_CLOCK ? FL_NO_BOX : FL_UP_BOX);
```

Note

From Duncan: (will be removed later, just for now as a reminder)
I've just added comments for the fl_color_chooser() functions, and in order to keep them and the general Function Reference information for them together, I created a new doxygen group, and used \ingroup in the three comment blocks. This creates a new Modules page (which may not be what we want) with links to it from the File Members and Fl_Color_Chooser.H pages. It needs a bit more experimentation on my part unless someone already knows how this should be handled. (Maybe we can add it to a functions.dox file that defines a functions group and do that for all of the function documentation?)

** Update**: the trick is not to create duplicate entries in a new group, but to move the function information into the doxygen comments for the class, and use the navigation links provided. Simply using \relatesalso as the first doxygen command in the function's comment puts it in the appropriate place. There is no need to have \defgroup and \ingroup as well, and indeed they don't work. So, to summarize:

** Gizmo.H **

```cpp
/** 
    \class Gizmo
    A gizmo that does everything
    */

    class Gizmo { etc
    }

    extern int popup_gizmo(...);
```

** Gizmo.cxx **

```cpp
/** \relatesalso Gizmo
    Pops up a gizmo dialog with a Gizmo in it
    */

    int popup_gizmo(...);
```
4.3 Developer Information

Comments Within Doxygen Comment Blocks

You can use HTML comment statements to embed comments in doxygen comment blocks. These comments will not be visible in the generated document.

<!-- *** This *** is *** invisible *** -->

This will be visible again.

The following text is a developer comment.

This will be shown as:

The following text is a developer comment.

Different Headlines

You can use HTML tags `<H1> ... <H4>` for headlines with different sizes. As of doxygen 1.8.x there must not be more than three spaces at the beginning of the line for this to work. Currently (doxygen 1.8.6) there seems to be no difference in the font sizes of `<H3>` and `<H4>` in the pdf output, whereas the html output uses different font sizes.

```
<H1>Headline in big text (H1)</H1>
<H2>Headline in big text (H2)</H2>
<H3>Headline in big text (H3)</H3>
<H4>Headline in big text (H4)</H4>
```

Headline in big text (H1)

Headline in big text (H2)

Headline in big text (H3)

Headline in big text (H4)

4.3.1 Non-ASCII Characters

Doxygen understands many HTML quoting characters like ", &uuml;, &ccedil; ı, but not all HTML quoting characters.

This will appear in the document:

For further informations about HTML quoting characters see

http://www.doxygen.org/manual/htmlcmds.html

Alternatively you can use UTF-8 encoding within Doxygen comments.

4.3.2 Document Structure

- \page creates a named page
- \section creates a named section within that page
- \subsection creates a named subsection within the current section
- \subsubsection creates a named subsubsection within the current subsection

All these statements take a "name" as their first argument, and a title as their second argument. The title can contain spaces.

The page, section, and subsection titles are formatted in blue color and a size like "<H1>" , "<H2>" , and "<H3>" , and "<H4>" , respectively.

By FLTK documentation convention, a file like this one with a doxygen documentation chapter has the name "<chapter>.dox". The \page statement at the top of the page is "\page <chapter> This is the title". Sections within a documentation page must be called "<chapter> _ <section>" , where "<chapter>" is the name part of the file, and "<section>" is a unique section name within the page that can be referenced in links. The same for subsections and subsubsections.

These doxygen page and section commands work only in special documentation chapters, not within normal source or header documentation blocks. However, links from normal (e.g. class) documentation to documentation sections do work.

Generated by Doxygen
4.3.3 Creating Links

Links to other documents and external links can be embedded with

- doxygen \ref links to other doxygen \page, \section, \subsection and \anchor locations
- HTML links without markup - doxygen creates "http://..." links automatically
- standard, non-Doxygen, HTML links
  - see chapter \ref unicode creates a link to the named chapter unicode that has been created with a \\page statement.
  - For further informations about quoting see http://www.doxygen.org/manual/htmlcmds.html
  - see <a href="https://www.fltk.org/">FLTK Library</a> creates a standard HTML link

appears as:

- see chapter Unicode and UTF-8 Support creates a link to the named chapter unicode that has been created with a \page statement.
- For further informations about quoting see http://www.doxygen.org/manual/htmlcmds.html
- see FLTK Library creates a standard HTML link

4.3.4 Paragraph Layout

There is no real need to use HTML <P> and </P> tags within the text to tell doxygen to start or stop a paragraph. In most cases, when doxygen encounters a blank line or some, but not all, \commands in the text it knows that it has reached the start or end of a paragraph. Doxygen also offers the \par command for special paragraph handling. It can be used to provide a paragraph title and also to indent a paragraph. Unfortunately \par won't do what you expect if you want to have doxygen links and sometimes html tags don't work either.

\par Normal Paragraph with title
This paragraph will have a title, but because there is a blank line between the \par and the text, it will have the normal layout.

\par Indented Paragraph with title
This paragraph will also have a title, but because there is no blank line between the \par and the text, it will be indented.

\par It is also possible to have an indented paragraph without title.
This is how you indent subsequent paragraphs.

\par No link to Fl_Widget::draw()
Note that the paragraph title is treated as plain text.
Doxygen type links will not work.
HTML characters and tags may or may not work.

Fl_Widget::draw() links and &quot;html&quot; tags work<br>
\par
Use a single line ending with <br> for complicated paragraph titles.

The above code produces the following paragraphs:
Normal Paragraph with title

This paragraph will have a title, but because there is a blank line between the `par` and the text, it will have the normal layout.

Indented Paragraph with title

This paragraph will also have a title, but because there is no blank line between the `par` and the text, it will be indented.

It is also possible to have an indented paragraph without title. This is how you indent subsequent paragraphs.

No link to Fl_Widget::draw()

Note that the paragraph title is treated as plain text. Doxygen type links will not work. HTML characters and tags may or may not work.

Fl_Widget::draw() links and "html" tags work

Use a single line ending with `<br>` for complicated paragraph titles.

4.3.5 Navigation Elements

Each introduction (tutorial) page ends with navigation elements. These elements must only be included in the html documentation, therefore they must be separated with `htmlonly` and `endhtmlonly`.

The following code gives the navigation bar at the bottom of this page:

\htmlonly
<br>
<table summary="navigation bar" width="100%" border="0">
<tr>
<td width="45%" align="LEFT">
<a class="el" href="migration_1_4.html">
[Prev]
Migrating Code from FLTK 1.3 to 1.4</a></td>
<td width="10%" align="CENTER">
<a class="el" href="index.html">
[Index]</a></td>
<td width="45%" align="RIGHT">
<a class="el" href="license.html">
[Next]
Software License</a></td>
</tr>
</table>
\endhtmlonly
Chapter 5

Todo List

Page Adding and Extending Widgets

Clarify Fl_Window::damage(uchar) handling - seems confused/wrong? ORing value doesn't match setting behavior in Fl_Widget.H!

Member Fl::now (double offset=0)

Fl::system_driver()->gettime() was implemented for the Forms library and has a limited resolution (on Windows: milliseconds). On POSIX platforms it uses gettimeofday() with microsecond resolution. A new function could use a better resolution on Windows with its multimedia timers which requires a new dependency: winmm.lib (dll). This could be a future improvement, maybe set as a build option or generally (requires Win95 or 98?).

Member Fl_Browser___::scrollbar_width () const

This method should eventually be removed in 1.4+

Member Fl_Browser___::scrollbar_width (int width)

This method should eventually be removed in 1.4+

Class Fl_Chart

Refactor Fl_Chart::type() information.

Member Fl_File_Input::errorcolor (Fl_Color c)

Remove Fl_File_Input::errorcolor(Fl_Color) in FLTK 1.5.0 or higher.

Member Fl_File_Input::errorcolor () const

Remove Fl_File_Input::errorcolor() in FLTK 1.5.0 or higher.

Member fl_filename_list (const char ∗d, struct dirent ∗∗∗l, Fl_File_Sort_F ∗s=fl_numericsort)

should support returning OS error messages

Class Fl_Grid

This (relative group coordinates of nested groups of Fl_Grid) needs explanation and maybe an example.

Member Fl_Grid::clear_layout ()

Fl_Grid::clear() needs to be implemented as documented above!

Member Fl_Grid::debug (int level=127)

Add more information about cells and children.

Control output by using level.

Member Fl_Grid::Fl_Grid (int X, int Y, int W, int H, const char ∗L=0)

More documentation of Fl_Grid constructor?

Member Fl_Grid::layout ()

Document when and why to call layout() w/o args. See Fl_Flex::layout()

Member Fl_Grid::layout (int rows, int cols, int margin=-1, int gap=-1)

Document when and why to call layout() w/o args. See Fl_Flex::layout()
Member **Fl_Group::delete_child** (int n)
Reimplementation of Fl_Group::delete_child(int) in more FLTK subclasses. This is not yet complete.

Member **fl_height** (int font, int size)
In the future, when the XFT issues are resolved, this function should simply return the 'size' value.

Member **Fl_Help_View::find** (const char ∗s, int p=0)
complex HTML entities for Unicode code points > 0x80 are currently treated like one byte (not character!) and do not (yet) match correctly ("&lt;" matches "<" but "&egrave;" doesn't match "é", and "ö" doesn't match "ö")

Member **Fl_Input_::handle_mouse** (int, int, int, int keepmark=0)
Add comment and parameters

Member **Fl_Input_::handletext** (int e, int, int, int, int)
Add comment and parameters

Struct **Fl_Label**
There is an aspiration that the Fl_Label type will become a widget by itself. That way we will be avoiding a lot of code duplication by handling labels in a similar fashion to widgets containing text. We also provide an easy interface for very complex labels, containing html or vector graphics. However, this re-factoring is not in place in this release.

Member **Fl_Menu_::add** (const char ∗s, int shortcut, Fl_Callback ∗, void ∗=0, int=0)
Raw integer shortcut needs examples. Dependent on responses to https://www.fltk.org/newsgroups.php?g=fltk.coredev+v:10086 and results of STR#2344

Class **Fl_Native_File_Chooser**
Improve documentation about selection of native file choosers on X11/Wayland.

Member **Fl_Shortcut**
Discuss and decide whether we can "shift" these special keyboard flags to the upper byte to enable full 21-bit Unicode characters (U+0000 .. U+10FFFF) plus the keyboard indicator bits as this was originally intended. This would be possible if we could rely on all programs being coded with symbolic names and not hard coded bit values.

Member **Fl_Terminal::scrollbar**
Support scrollbar_left/right() - See Fl_Browser_::scrollbar docs

Member **Fl_Text_Display::extend_range_for_styles** (int ∗start, int ∗end)
Unicode?

Member **Fl_Text_Display::handle_vline** (int mode, int lineStart, int lineLen, int leftChar, int rightChar, int topClip, int bottomClip, int leftClip, int rightClip) const
we need to handle hidden hyphens and tabs here!
we handle all styles and selections
we must provide code to get pixel positions of the middle of a character as well

Member **Fl_Text_Display::overstrike** (const char ∗text)
Unicode? Find out exactly what we do here and simplify.

Member **Fl_Text_Display::position_to_linecol** (int pos, int ∗lineNum, int ∗column) const
a column number makes little sense in the UTF-8/variable font width environment. We will have to further define what exactly we want to return. Please check the functions that call this particular function.

Member **Fl_Text_Display::scroll** (int topLineNum, int horizOffset)
Column numbers make little sense here.

Member **Fl_Text_Display::scrollbar_width** (int width)
This method should eventually be removed

Member **Fl_Text_Display::scrollbar_width()** const
This method should be removed.
Member `Fl_Text_Display::shortcut (int s)`

FIXME : get set methods pointing on shortcut_ have no effects as shortcut_ is unused in this class and derived!

Member `Fl_Text_Display::shortcut () const`

FIXME : get set methods pointing on shortcut_ have no effects as shortcut_ is unused in this class and derived!

Member `Fl_Text_Display::wrapped_column (int row, int column) const`

What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)

Unicode?

Member `Fl_Text_Display::wrapped_row (int row) const`

What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)

Member `Fl_Tiled_Image::Fl_Tiled_Image (Fl_Image ∗i, int W=0, int H=0)`

Fix Fl_Tiled_Image as background image for widgets and windows and fix the implementation of Fl::scheme(const char ∗).

Member `Fl_Tree::handle (int e) FL_OVERRIDE`

add Fl_Widget_Tracker (see Fl_Browser_.cxx::handle())

Member `Fl_Tree::is_scrollbar (Fl_Widget ∗w)`

should be const

Member `Fl_Tree::show_self ()`

should be const

Member `Fl_Window::show () FL_OVERRIDE`

Check if we can remove resetting the current group in a later FLTK version (after 1.3.x). This may break "already broken" programs though if they rely on this "feature".

Page FLTK Basics

This section needs a major rework. Add a chapter "Building FLTK with CMake".

Page Handling Events

Add details on how to detect repeating keys, since on some X servers a repeating key will generate both FL←KEYUP and FL_KEYDOWN, such that to tell if a key is held, you need Fl::event_key(int) to detect if the key is being held down during FL_KEYUP or not.

Page Unicode and UTF-8 Support

Verify 16/24 bit Unicode limit for different character sets? OksiD's code appears limited to 16-bit whereas the FLTK2 code appears to handle a wider set. What about illegal characters? See comments in fl_utf8fromwc() and fl_ut8toUtf16().

Work through the code and this documentation to harmonize the [OksiD] and [fltk2] functions.

FLTK 1.3 and later supports the full Unicode range (21 bits), but there are a few exceptions, for instance binary shortcut values in menus (Fl_Shortcut) can only be used with characters from the BMP (16 bits). This may be extended in a future FLTK version.
Chapter 6

Deprecated List

Member Fl::release ()
Use Fl::grab(0) instead.

Member Fl::set_idle (FI_Old_Idle_Handler cb)
This method is obsolete - use the add_idle() method instead.

Member Fl::version ()
Use int Fl::api_version() instead.

Member fl_ask (const char ∗fmt,...)
fl_ask() is deprecated since it uses "Yes" and "No" for the buttons which does not conform to the current FLTK Human Interface Guidelines. Use fl_choice() with the appropriate verbs instead.

Member Fl_Browser::::position () const
"in 1.4.0 - use vposition() instead"

Member Fl_Browser::::position (int pos)
"in 1.4.0 - use vposition(pos) instead"

Member Fl_Browser::::scrollbar_width () const
Use scrollbar_size() instead.

Member Fl_Browser::::scrollbar_width (int width)
Use scrollbar_size() instead.

Member fl_clip (int x, int y, int w, int h)
Please use fl_push_clip(int x, int y, int w, int h) instead. fl_clip(int, int, int, int) will be removed in FLTK 1.5.

Member Fl_File_Input::::errorcolor () const
Will be removed in FLTK 1.5.0 or higher.

Member Fl_File_Input::::errorcolor (Fl_Color c)
Will be removed in FLTK 1.5.0 or higher.

Member fl_find (Window xid)
Kept in the X11, Windows, and macOS platforms for compatibility with FLTK versions before 1.4. Please use fl_x11_find(Window), fl_wl_find(struct wid_window*), fl_win32_find(HWND) or fl_mac_find(FLWindow*) with FLTK 1.4.0 and above.

Member Fl_GIF_Image::::GIF Image (const char ∗imagename, const unsigned char ∗data)
Please use Fl_GIF_Image(const char ∗imagename, const unsigned char ∗data, const size_t length) instead.

Member Fl_Group::::focus (Fl_Widget ∗W)
This is for backwards compatibility only. You should use W->take_focus() instead.

Member Fl_Group::::sizes ()
Deprecate since 1.4.0. Please use bounds() instead.
Member `Fl_Image::draw_scaled` (int X, int Y, int W, int H)
   Only for API compatibility with FLTK 1.3.4.

Member `Fl_Image::label (Fl_Menu_Item ∗m)`
   Please use `Fl_Menu_Item::image()` instead.

Member `Fl_Image::label (Fl_Widget ∗w)`
   Please use `Fl_Widget::image()` or `Fl_Widget::deimage()` instead.

Member `Fl_Image_Surface::highres_image ()`
   Use `image()` instead.

Member `Fl_Input::position (int p)`
   "in 1.4.0 - use insert_position(p) instead"

Member `Fl_Input::position (int p, int m)`
   "in 1.4.0 - use insert_position(p, m) or Fl_Widget::position(x, y) instead"

Member `Fl_Input::position () const`
   "in 1.4.0 - use insert_position() instead"

Member `Fl_Menu_Item::check ()`
   Please use `Fl_Menu_Item::set()` instead. This method will be removed in FLTK 1.5.0 or later.

Member `Fl_Menu_Item::checked () const`
   Please use `Fl_Menu_Item::value()` instead. This method will be removed in FLTK 1.5.0 or later.

Member `Fl_Menu_Item::uncheck ()`
   Please use `Fl_Menu_Item::clear()` instead. This method will be removed in FLTK 1.5.0 or later.

Member `Fl_Multi_Label::label (Fl_Menu_Item ∗m)`
   since 1.4.0: please use `Fl_Menu_Item::label(Fl_Multi_Label ∗)`

Member `Fl_Preferences::Fl_Preferences (const char ∗path, const char ∗vendor, const char ∗application)`
   "in 1.4.0 - use Fl_Preferences(path, vendor, application, flags) instead"

Member `Fl_Text_Display::scrollbar_width (int width)`
   Use `scrollbar_size()` instead.

Member `Fl_Text_Display::scrollbar_width () const`
   Use `scrollbar_size()` instead.

Member `Fl_Text_Selection::position (int ∗startpos, int ∗endpos) const`
   "in 1.4.0 - use selected(startpos, endpos) instead"

Member `Fl_Tile::position (int oldx, int oldy, int newx, int newy)`
   "in 1.4.0 - use move_intersection(p) instead"

Member `Fl_Tree::first_visible ()`
   in 1.3.3 ABI – use `first_visible_item()` instead.

Member `Fl_Tree::item_clicked (Fl_Tree_Item ∗val)`
   in 1.3.3 ABI – use `callback_item()` instead.

Member `Fl_Tree::item_clicked ()`
   in 1.3.3 ABI – use `callback_item()` instead.

Member `Fl_Tree::last_visible ()`
   in 1.3.3 – use `last_visible_item()` instead.
**Member Fl_Tree_Item::Fl_Tree_Item (const Fl_Tree_Prefs &prefs)**

in 1.3.3 ABI – you must use Fl_Tree_Item(Fl_Tree+) for proper horizontal scrollbar behavior.

**Member Fl_Tree_Item::next_displayed (Fl_Tree_Prefs &prefs)**

in 1.3.3 for confusing name, use next_visible() instead

**Member Fl_Tree_Item::prev_displayed (Fl_Tree_Prefs &prefs)**

in 1.3.3 for confusing name, use prev_visible()

**Member FL_VERSION**

This double version number is retained for compatibility with existing program code. New code should use int FL_API_VERSION instead. FL_VERSION is deprecated because comparisons of floating point values may fail due to rounding errors. However, there are currently no plans to remove this deprecated constant.

**Member Fl_Widget::color2 (unsigned a)**

Use selection_color(unsigned) instead.

**Member Fl_Widget::color2 () const**

Use selection_color() instead.

**Member Fl_Window::free_position ()**

please use force_position(0) instead

**Member Fl_Window::icon () const**

in 1.3.3

**Member Fl_Window::icon (const void *ic)**

in 1.3.3 in favor of platform-independent methods Fl_Window::icon(const Fl_RGB_Image *icon) and Fl_Window::icons(const Fl_RGB_Image *icons[], int count).

**Page Operating System Issues**

Kept for compatibility with FLTK versions before 1.4. Use preferentially fl_x11_xid(const Fl_Window *) with versions 1.4 and above.

Kept for compatibility with FLTK versions before 1.4. Use preferentially fl_x11_find(Window) with versions 1.4 and above.
Chapter 7

Topic Index

7.1 Topics

Here is a list of all topics with brief descriptions:

- Callback Function Typedefs .......................................................... 265
- Windows handling functions ......................................................... 266
- Events handling functions ............................................................. 268
- Selection & Clipboard functions ................................................... 282
- Screen functions ........................................................................ 287
- Color & Font functions ............................................................... 292
- Drawing functions ..................................................................... 306
- Multithreading support functions ................................................ 338
- Safe widget deletion support functions ...................................... 340
- Cairo Support Functions and Classes .......................................... 342
- Unicode and UTF-8 functions ..................................................... 345
- String handling functions .......................................................... 361
- Mac OS X-specific symbols ......................................................... 361
- Common Dialog Classes and Functions ........................................ 363
- The gtk_shell1 interface ............................................................. 377
- The gtk_surface1 interface ......................................................... 378
- File names and URI utility functions .......................................... 378
Chapter 8

Hierarchical Index

8.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- Fl_Grid::Cell .................................................. 387
- Fl_Terminal::CharStyle .................................... 387
- Fl_GIF_Image::GIF_FRAME::CPAL .......................... 388
- Fl_Terminal::Cursor ......................................... 388
- Fl_Preferences::Entry ........................................ 389
- Fl_Terminal::EscapeSeq .................................... 389
- Fl ................................................................. 390
- Fl_Cairo_State ............................................... 529
- Fl_Callback_User_Data ..................................... 543
- FL_CHART_ENTRY .............................................. 557
- Fl_End ......................................................... 663
- Fl_File_Chooser ............................................. 682
- Fl_File_Icon .................................................. 689
- Fl_Gl_Choice .................................................. 784
- Fl_Glut_Bitmap_Font ....................................... 805
- Fl_Glut_StrokeChar ......................................... 806
- Fl_Glut_StrokeFont .......................................... 806
- Fl_Glut_StrokeStrip ......................................... 806
- Fl_Glut_StrokeVertex ....................................... 806
- Fl_Help_Block ............................................... 862
- Fl_Help_Dialog ............................................... 863
- Fl_Help_Font_Stack ......................................... 865
- Fl_Help_Font_Style .......................................... 865
- Fl_Help_Link .................................................. 866
- Fl_Help_Target ............................................... 866
- Fl_Image ......................................................... 932
  - Fl_Bitmap .................................................. 444
    - Fl_XBM_Image ........................................... 1949
  - Fl_Pixmap ................................................ 1192
    - Fl_GIF_Image ........................................... 779
      - Fl_Anim_GIF_Image ..................................... 429
    - Fl_XPM_Image ........................................... 1952
  - Fl_RGB_Image ............................................. 1308
  - Fl_BMP_Image ............................................. 450
    - Fl_ICO_Image ........................................... 928
  - Fl_JPEG_Image ............................................ 1009
  - Fl_PNG_Image ............................................. 1204
  - Fl_PNM_Image ............................................. 1208
  - Fl_SVG_Image ............................................. 1480
  - Fl_Shared_Image ......................................... 1401
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Tiled_Image</td>
<td>1713</td>
</tr>
<tr>
<td>Fl_Image_Reader</td>
<td>941</td>
</tr>
<tr>
<td>Fl_Label</td>
<td>1013</td>
</tr>
<tr>
<td>Fl_Mac_App_Menu</td>
<td>1031</td>
</tr>
<tr>
<td>Fl_Menu_Item</td>
<td>1076</td>
</tr>
<tr>
<td>Fl_Multi_Label</td>
<td>1115</td>
</tr>
<tr>
<td>Fl_Native_File_Chooser</td>
<td>1137</td>
</tr>
<tr>
<td>Fl_Plugin</td>
<td>1197</td>
</tr>
<tr>
<td>Fl_Device_Plugin</td>
<td>1197</td>
</tr>
<tr>
<td>Fl_Preferences</td>
<td>1228</td>
</tr>
<tr>
<td>Fl_Rect</td>
<td>1288</td>
</tr>
<tr>
<td>Fl_Scroll::Fl_Region_LRTB</td>
<td>1290</td>
</tr>
<tr>
<td>Fl_Scroll::Fl_Region_XYWH</td>
<td>1290</td>
</tr>
<tr>
<td>Fl_Scheme</td>
<td>1339</td>
</tr>
<tr>
<td>Fl_Scroll::Fl_Scrollbar_Data</td>
<td>1377</td>
</tr>
<tr>
<td>Fl_Surface_Device</td>
<td>1474</td>
</tr>
<tr>
<td>Fl_Display_Device</td>
<td>1474</td>
</tr>
<tr>
<td>Fl_Widget_Surface</td>
<td>1905</td>
</tr>
<tr>
<td>Fl_Copy_Surface</td>
<td>623</td>
</tr>
<tr>
<td>Fl_EPS_File_Surface</td>
<td>664</td>
</tr>
<tr>
<td>Fl_Image_Surface</td>
<td>942</td>
</tr>
<tr>
<td>Fl_Paged_Device</td>
<td>1186</td>
</tr>
<tr>
<td>Fl_PostScript_File_Device</td>
<td>1219</td>
</tr>
<tr>
<td>Fl_Printer</td>
<td>1248</td>
</tr>
<tr>
<td>Fl_SVG_File_Surface</td>
<td>1476</td>
</tr>
<tr>
<td>Fl_Text_Buffer</td>
<td>1604</td>
</tr>
<tr>
<td>Fl_Text_Selection</td>
<td>1692</td>
</tr>
<tr>
<td>Fl_Timeout</td>
<td>1717</td>
</tr>
<tr>
<td>Fl_Tooltip</td>
<td>1740</td>
</tr>
<tr>
<td>Fl_Tree_Item</td>
<td>1793</td>
</tr>
<tr>
<td>Fl_Tree_Item_Array</td>
<td>1815</td>
</tr>
<tr>
<td>Fl_Tree_Prefs</td>
<td>1818</td>
</tr>
<tr>
<td>Fl_Widget</td>
<td>1865</td>
</tr>
<tr>
<td>Fl_Box</td>
<td>454</td>
</tr>
<tr>
<td>Fl_Button</td>
<td>518</td>
</tr>
<tr>
<td>Fl_Light_Button</td>
<td>1014</td>
</tr>
<tr>
<td>Fl_Check_Button</td>
<td>572</td>
</tr>
<tr>
<td>Fl_Radio_Light_Button</td>
<td>1272</td>
</tr>
<tr>
<td>Fl_Round_Button</td>
<td>1324</td>
</tr>
<tr>
<td>Fl_Radio_Round_Button</td>
<td>1280</td>
</tr>
<tr>
<td>Fl_Radio_Button</td>
<td>1265</td>
</tr>
<tr>
<td>Fl_Repeat_Button</td>
<td>1291</td>
</tr>
<tr>
<td>Fl_Return_Button</td>
<td>1299</td>
</tr>
<tr>
<td>Fl_Shortcut_Button</td>
<td>1412</td>
</tr>
<tr>
<td>Fl_Toggle_Button</td>
<td>1732</td>
</tr>
<tr>
<td>Fl_Chart</td>
<td>544</td>
</tr>
<tr>
<td>Fl_Clock_Output</td>
<td>601</td>
</tr>
<tr>
<td>Fl_Clock</td>
<td>592</td>
</tr>
<tr>
<td>Fl_Round_Clock</td>
<td>1332</td>
</tr>
<tr>
<td>Fl_FormsBitmap</td>
<td>750</td>
</tr>
<tr>
<td>Fl_FormsPixmap</td>
<td>757</td>
</tr>
<tr>
<td>Fl_FormsText</td>
<td>764</td>
</tr>
<tr>
<td>Fl_Free</td>
<td>771</td>
</tr>
<tr>
<td>Fl_Group</td>
<td>843</td>
</tr>
<tr>
<td>Fl_Browser</td>
<td>494</td>
</tr>
</tbody>
</table>
8.1 Class Hierarchy

Fl_Browser .......................................................... 462
Fl_File_Browser .................................................. 667
Fl_Hold_Browser ................................................... 881
Fl_Multi_Browser .................................................. 1101
Fl_Select_Browser .................................................. 1388
Fl_Check_Browser .................................................. 557
Fl_Color_Chooser ................................................... 611
Fl_Flex ............................................................. 722
Fl_Grid ............................................................... 822
Fl_Help_View ....................................................... 867
Fl_Input_Choice .................................................... 987
Fl_Pack .............................................................. 1176
Fl_Scroll ............................................................. 1351
Fl_Spinner ........................................................... 1451
Fl_Table .............................................................. 1503
Fl_Table_Row ........................................................ 1529
Fl_Tabs ............................................................... 1544
Fl_Terminal .......................................................... 1565
Fl_Text_Display ..................................................... 1622
Fl_Text_Editor ....................................................... 1669
Fl_Tile ................................................................. 1695
Fl_Tree ............................................................... 1745
Fl_Window ............................................................. 1910
Fl_Double_Window .................................................. 650
Fl_Cairo_Window .................................................... 530
Fl_Overlay_Window ................................................ 1162
Fl_GL_Window ........................................................ 785
Fl_Glut_Window ...................................................... 807
Fl_Single_Window ................................................... 1429
Fl_Menu_Window ....................................................... 1089
Fl_Wizard ............................................................. 1940
Fl_Input .............................................................. 960
Fl_File_Input ........................................................ 694
Fl_Float_Input ....................................................... 740
Fl_Int_Input .......................................................... 999
Fl_Multiline_Input ................................................ 1117
Fl_Output .............................................................. 1152
Fl_Multiline_Output ............................................... 1127
Fl_Secret_Input ..................................................... 1377
Fl_Spinner::Fl_Spinner_Input .................................. 1464
Fl_Menu ............................................................. 1032
Fl_Choice .............................................................. 580
Fl_Scheme_Choice .................................................. 1341
Fl_Menu_Bar .......................................................... 1054
Fl_Sys_Menu_Bar ................................................... 1487
Fl_Menu_Button ..................................................... 1064
Fl_Positioner ....................................................... 1211
Fl_Progress ........................................................ 1257
Fl_Timer .............................................................. 1724
Fl_Valuator .......................................................... 1823
Fl_Adjuster ........................................................... 420
Fl_Counter ............................................................ 627
Fl_Simple_Counter ................................................ 1421
Fl_Dial ............................................................... 639
Fl_Fill_Dial ........................................................... 706
Fl_Line_Dial ............................................................. 1023
Fl_Roller ................................................................. 1315
Fl_Slider ................................................................. 1441
  Fl_Fill_Slider ......................................................... 714
  Fl_Hor_Fill_Slider ..................................................... 894
  Fl_Hor_Nice_Slider ..................................................... 903
  Fl_Hor_Slider .......................................................... 911
  Fl_Nice_Slider .......................................................... 914
  Fl_Scrollbar .......................................................... 914
  Fl_Value_Slider ........................................................ 1855
    Fl_Hor_Value_Slider ................................................. 919
Fl_Value_Input ........................................................... 1833
Fl_Value_Output .......................................................... 1845
Fl_Widget_Tracker ....................................................... 1909
Fl_XColor ............................................................... 1952
Fl_GIF_Image::GIF_FRAME ............................................... 1955
gtk_shell1_listener ..................................................... 1956
gtk_surface1_listener ..................................................... 1956
Fl_ICO_Image::IconDirEntry ........................................... 1956
Fl_Text_Editor::Key_Binding .......................................... 1957
Fl_Terminal::Margin ..................................................... 1957
Fl_Preferences::Name ..................................................... 1957
Fl_Preferences::Node ..................................................... 1958
Fl_Paged_Device::page_format ......................................... 1959
Fl_Terminal::PartialUtf8Buf .......................................... 1960
Fl_Terminal::RingBuffer ............................................... 1960
Fl_Preferences::RootNode ............................................... 1961
Fl_Scroll::ScrollInfo ................................................... 1961
Fl_Terminal::Selection ................................................ 1962
Fl_Tile::Size_Range ..................................................... 1963
Fl_Text_Display::Style_Table_Entry ................................... 1963
Fl_Terminal::Utf8Char .................................................. 1963
Chapter 9

Class Index

9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

- Fl_Grid::Cell
- Fl_Terminal::CharStyle
- Fl_GIF_Image::GIF_FRAME::CPAL
- Fl_Terminal::Cursor
- Fl_Preferences::Entry
- Fl_Terminal::EscapeSeq

The Fl is the FLTK global (static) class containing state information and global methods for the current application.

- Fl_Adjuster
  - Was stolen from Prisms, and has proven to be very useful for values that need a large dynamic range.

- Fl_Anim_GIF_Image
  - Supports loading, caching, and drawing of animated Compuserve GIF℠ images.

- Fl_Bitmap
  - Supports caching and drawing of mono-color (bitmap) images.

- Fl_BMP_Image
  - Supports loading, caching, and drawing of Windows Bitmap (BMP) image files.

- Fl_Box
  - This widget simply draws its box, and possibly its label.

- Fl_Browser
  - Displays a scrolling list of text lines, and manages all the storage for the text.

- Fl_Browser
  - This is the base class for browsers.

- Fl_Button
  - Buttons generate callbacks when they are clicked by the user.

- Fl_Cairo_State
  - Contains all the necessary info on the current cairo context.

- Fl_Cairo_Window
  - This defines an FLTK window with Cairo support.

- Fl_Callback_User_Data
  - A class prototype that allows for additional data in callbacks.

- Fl_Chart
  - Fl_Chart displays simple charts.

- FL_CHART_ENTRY
  - For internal use only.

- Fl_Check_Browser
  - Displays a scrolling list of text lines that may be selected and/or checked by the user.

- Fl_Check_Button
  - A button with a "checkmark" to show its status.

Generated by Doxygen
Fl_Choice
A button that is used to pop up a menu .................................................. 580

Fl_Clock
This widget provides a round analog clock display .................................... 592

Fl_Clock_Output
This widget can be used to display a program-supplied time ....................... 601

Fl_Color_Chooser
Standard RGB color chooser ................................................................. 611

Fl_Copy_Surface
Supports copying of graphical data to the clipboard .................................. 623

Fl_Counter
Controls a single floating point value with button (or keyboard) arrows ........ 627

Fl_Device_Plugin
This plugin socket allows the integration of new device drivers for special window or screen types 638

Fl_Dial
Circular dial to control a single floating point value ............................... 639

Fl_Display_Device
The computer's display ........................................................................... 648

Fl_Double_Window
The Fl_Double_Window provides a double-buffered window ....................... 650

Fl_End
This is a dummy class that allows you to end a Fl_Group in a constructor list of a class: 663

Fl_EPS_File_Surface
Encapsulated PostScript drawing surface .................................................. 664

Fl_File_Browser
Displays a list of filenames, optionally with file-specific icons .................... 667

Fl_File_Chooser
Displays a standard file selection dialog that supports various selection modes 682

Fl_File_Icon
Manages icon images that can be used as labels in other widgets and as icons in the FileBrowser widget ................................. 689

Fl_File_Input
This widget displays a pathname in a text input field ............................... 694

Fl_Fill_Dial
Draws a dial with a filled arc ................................................................. 706

Fl_Fill_Slider
Widget that draws a filled horizontal slider, useful as a progress or value meter 714

Fl_Flex
Fl_Flex is a container (layout) widget for one row or one column of widgets .......... 722

Fl_Float_Input
Subclass of Fl_Input that only allows the user to type floating point numbers (sign, digits, decimal point, more digits, 'E' or 'e', sign, digits) 740

Fl_FormsBitmap
Forms compatibility Bitmap Image Widget ............................................ 750

Fl_FormsPixmap
Forms pixmap drawing routines ........................................................... 757

Fl_FormsText
............................................................... 764

Fl_Free
Emulation of the Forms "free" widget ..................................................... 771

Fl_GIF_Image
Supports loading, caching, and drawing of Compuserve GIF^SM^ images .......... 779

Fl_GL_Choice
............................................................... 784

Fl_GL_Window
Sets things up so OpenGL works ......................................................... 785

Fl_Gl_Choice
............................................................... 784

Fl_Gl_Window
Sets things up so OpenGL works ......................................................... 785

Fl_Gl_Choice
............................................................... 784

Fl_Gl_Window
Sets things up so OpenGL works ......................................................... 785

Fl_Glut_Bitmap_Font
Fltk glut font/size attributes used in the glutXXX functions .................... 805

Fl_Glut_StrokeChar
............................................................... 806

Fl_Glut_StrokeFont
............................................................... 806

Generated by Doxygen
**Fl_Glut_StrokeStrip**

GLUT is emulated using this window class and these static variables (plus several more static variables hidden in glut_compatibility.cxx):

**Fl_Glut_StrokeVertex**

**Fl_Glut_Window**

**Fl_Grid**

Fl_Grid is a container (layout) widget with multiple columns and rows.

**Fl_Group**

FLTK container widget.

**Fl_Help_Block**

Definition of a link for the html viewer.

**Fl_Help_Dialog**

Displays a standard help dialog window using the Fl_Help_View widget.

**Fl_Help_Font_Stack**

Fl_Help_View font stack element definition.

**Fl_Help_Font_Style**

Fl_Help_Font_Stack.

**Fl_Help_Link**

Definition of a link for the html viewer.

**Fl_Help_Target**

Fl_Help_Target structure.

**Fl_Help_View**

Displays HTML text.

**Fl_Hold_Browser**

The Fl_Hold_Browser is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space.

**Fl_Hor_Fill_Slider**

Single thumb tab slider.

**Fl_Hor_Nice_Slider**

Horizontal Slider class.

**Fl_Hor_Slider**

Horizontal Slider class.

**Fl_ICO_Image**

Supports loading, caching, and drawing of Windows icon (.ico) files.

**Fl_Image**

Base class for image caching, scaling and drawing.

**Fl_Image_Reader**

Directs all graphics requests to an Fl_Image.

**Fl_Input**

This is the FLTK text input widget.

**Fl_Input_Choice**

A combination of the input widget and a menu button.

**Fl_Int_Input**

Subclass of Fl_Input that only allows the user to type decimal digits (or hex numbers of the form 0xaff).

**Fl_JPEG_Image**

Supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG) File Interchange Format (JFIF) images.

**Fl_Label**

This struct stores all information for a text or mixed graphics label.

**Fl_Light_Button**

This subclass displays the "on" state by turning on a light, rather than drawing pushed in.

**Fl_Line_Dial**

Base class of all widgets that have a menu in FLTK.
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Menu_Bar</td>
<td>This widget provides a standard menubar interface</td>
<td>1054</td>
</tr>
<tr>
<td>Fl_Menu_Button</td>
<td>This is a button that when pushed pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects</td>
<td>1064</td>
</tr>
<tr>
<td>Fl_Menu_Item</td>
<td>The Fl_Menu_Item structure defines a single menu item that is used by the Fl_Menu_class</td>
<td>1076</td>
</tr>
<tr>
<td>Fl_Menu_Window</td>
<td>Window type used for menus</td>
<td>1089</td>
</tr>
<tr>
<td>Fl_Multi_Browser</td>
<td>Subclass of Fl_Browser which lets the user select any set of the lines</td>
<td>1101</td>
</tr>
<tr>
<td>Fl_Multi_Label</td>
<td>Allows a mixed text and/or graphics label to be applied to an Fl_Menu_Item or Fl_Widget</td>
<td>1115</td>
</tr>
<tr>
<td>Fl_Multiline_Input</td>
<td>This input field displays \n characters as new lines rather than ^J, and accepts the Return, Tab, and up and down arrow keys</td>
<td>1117</td>
</tr>
<tr>
<td>Fl_Multiline_Output</td>
<td>This widget is a subclass of Fl_Output that displays multiple lines of text</td>
<td>1127</td>
</tr>
<tr>
<td>Fl_Native_File_Chooser</td>
<td>This class lets an FLTK application easily and consistently access the operating system's native file chooser</td>
<td>1137</td>
</tr>
<tr>
<td>Fl_Nice_Slider</td>
<td></td>
<td>1144</td>
</tr>
<tr>
<td>Fl_Output</td>
<td>This widget displays a piece of text</td>
<td>1152</td>
</tr>
<tr>
<td>Fl_Overlay_Window</td>
<td>This window provides double buffering and also the ability to draw the &quot;overlay&quot; which is another picture placed on top of the main image</td>
<td>1162</td>
</tr>
<tr>
<td>Fl_Pack</td>
<td></td>
<td>1176</td>
</tr>
<tr>
<td>Fl_Paged_Device</td>
<td></td>
<td>1186</td>
</tr>
<tr>
<td>Fl_Pixmap</td>
<td>Represents page-structured drawing surfaces</td>
<td>1192</td>
</tr>
<tr>
<td>Fl_PNG_Image</td>
<td>Supports loading, caching, and drawing of Portable Network Graphics (PNG) image files</td>
<td>1204</td>
</tr>
<tr>
<td>Fl_PNM_Image</td>
<td>Supports loading, caching, and drawing of Portable Anymap (PNM, PBM, PGM, PPM) image files</td>
<td>1208</td>
</tr>
<tr>
<td>Fl_Positioner</td>
<td>This class is provided for Forms compatibility</td>
<td>1211</td>
</tr>
<tr>
<td>Fl_PostScript_File_Device</td>
<td>To send graphical output to a PostScript file</td>
<td>1219</td>
</tr>
<tr>
<td>Fl_Preferences</td>
<td></td>
<td>1228</td>
</tr>
<tr>
<td>Fl_Printer</td>
<td>OS-independent print support</td>
<td>1248</td>
</tr>
<tr>
<td>Fl_Progress</td>
<td>Displays a progress bar for the user</td>
<td>1257</td>
</tr>
<tr>
<td>Fl_Radio_Button</td>
<td></td>
<td>1265</td>
</tr>
<tr>
<td>Fl_Radio_Light_Button</td>
<td></td>
<td>1272</td>
</tr>
<tr>
<td>Fl_Radio_Round_Button</td>
<td></td>
<td>1280</td>
</tr>
<tr>
<td>Fl_Rect</td>
<td>Rectangle with standard FLTK coordinates (X, Y, W, H)</td>
<td>1288</td>
</tr>
<tr>
<td>Fl_Scroll::Fl_Region_LRTB</td>
<td>A local struct to manage a region defined by left/right/top/bottom</td>
<td>1290</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Fl_Scroll::Fl_Region_XYWH</td>
<td>A local struct to manage a region defined by xywh</td>
<td>1290</td>
</tr>
<tr>
<td>Fl_Repeat_Button</td>
<td>The Fl_Repeat_Button is a subclass of Fl_Button that generates a callback when it is pressed and then repeatedly generates callbacks as long as it is held down</td>
<td>1291</td>
</tr>
<tr>
<td>Fl_Return_Button</td>
<td>The Fl_Return_Button is a subclass of Fl_Button that generates a callback when it is pressed or when the user presses the Enter key</td>
<td>1299</td>
</tr>
<tr>
<td>Fl_RGB_Image</td>
<td>Supports caching and drawing of full-color images with 1 to 4 channels of color information</td>
<td>1308</td>
</tr>
<tr>
<td>Fl_Roller</td>
<td>&quot;dolly&quot; control commonly used to move 3D objects</td>
<td>1315</td>
</tr>
<tr>
<td>Fl_Round_Button</td>
<td>Buttons generate callbacks when they are clicked by the user</td>
<td>1324</td>
</tr>
<tr>
<td>Fl_Round_Clock</td>
<td>A clock widget of type FL_ROUND_CLOCK</td>
<td>1332</td>
</tr>
<tr>
<td>Fl_Scheme</td>
<td></td>
<td>1339</td>
</tr>
<tr>
<td>Fl_Scheme_Choice</td>
<td></td>
<td>1341</td>
</tr>
<tr>
<td>Fl_Scroll</td>
<td>This container widget lets you maneuver around a set of widgets much larger than your window</td>
<td>1351</td>
</tr>
<tr>
<td>Fl_Scrollbar</td>
<td>Displays a slider with arrow buttons at the ends of the scrollbar</td>
<td>1366</td>
</tr>
<tr>
<td>Fl_Scroll::Fl_Scrollbar_Data</td>
<td>A local struct to manage a scrollbar's xywh region and tab values</td>
<td>1377</td>
</tr>
<tr>
<td>Fl_Secret_Input</td>
<td>Subclass of Fl_Input that displays its input as a string of placeholders</td>
<td>1377</td>
</tr>
<tr>
<td>Fl_Select_Browser</td>
<td>The class is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space</td>
<td>1388</td>
</tr>
<tr>
<td>Fl_Shared_Image</td>
<td>This class supports caching, loading, and drawing of image files</td>
<td>1401</td>
</tr>
<tr>
<td>Fl_Shortcut_Button</td>
<td>A button that allows the user to type a key combination to create shortcuts</td>
<td>1412</td>
</tr>
<tr>
<td>Fl_Simple_Counter</td>
<td>This widget creates a counter with only 2 arrow buttons</td>
<td>1421</td>
</tr>
<tr>
<td>Fl_Single_Window</td>
<td>This is the same as Fl_Window</td>
<td>1429</td>
</tr>
<tr>
<td>Fl_Slider</td>
<td>Sliding knob inside a box</td>
<td>1441</td>
</tr>
<tr>
<td>Fl_Spinner</td>
<td>This widget is a combination of a numerical input widget and repeat buttons</td>
<td>1451</td>
</tr>
<tr>
<td>Fl_Spinner::Fl_Spinner_Input</td>
<td></td>
<td>1464</td>
</tr>
<tr>
<td>Fl_Surface_Device</td>
<td>A drawing surface that's susceptible to receive graphical output</td>
<td>1474</td>
</tr>
<tr>
<td>Fl_SVG_File_Surface</td>
<td>A drawing surface producing a Scalable Vector Graphics (SVG) file</td>
<td>1476</td>
</tr>
<tr>
<td>Fl_SVG_Image</td>
<td>Supports loading, caching and drawing of scalable vector graphics (SVG) images</td>
<td>1480</td>
</tr>
<tr>
<td>Fl_Sys_Menu_Bar</td>
<td>A class to create and modify menus that appear on macOS in the menu bar at the top of the screen</td>
<td>1487</td>
</tr>
<tr>
<td>Fl_Table</td>
<td>A table of widgets or other content</td>
<td>1503</td>
</tr>
<tr>
<td>Fl_Table_Row</td>
<td>A table with row selection capabilities</td>
<td>1529</td>
</tr>
<tr>
<td>Fl_Tabs</td>
<td>Container widget that displays a set of tabs, with each tab representing a different child widget</td>
<td>1544</td>
</tr>
</tbody>
</table>
FL_Terminal
Terminal widget supporting Unicode/utf-8, ANSI/xterm escape codes with full RGB color control

FL_Text_Buffer
This class manages Unicode text displayed in one or more FL_Text_Display widgets

FL_Text_Display
Rich text display widget

FL_Text_Editor
This is the FLTK text editor widget

FL_Text_Selection
This is an internal class for FL_Text_Buffer to manage text selections

FL_Tile
Lets you resize its children by dragging the border between them

FL_Text_Buffer
This class manages Unicode text displayed in one or more FL_Text_Display widgets

FL_Terminal
Terminal widget supporting Unicode/utf-8, ANSI/xterm escape codes with full RGB color control

FL_Text_Display
Rich text display widget

FL_Text_Editor
This is the FLTK text editor widget

FL_Text_Selection
This is an internal class for FL_Text_Buffer to manage text selections

FL_Tile
Lets you resize its children by dragging the border between them

FL_Timeout
The internal class FL_Timeout handles all timeout related functions

FL_Timer
This is provided only to emulate the Forms Timer widget

FL_Toggle_Button
The toggle button is a push button that needs to be clicked once to toggle on, and one more time
to toggle off

FL_Tooltip
Tooltip support for all FLTK widgets

FL_Tree
Tree widget

FL_Tree_Item
Tree widget item

FL_Tree_Item_Array
Manages an array of FL_Tree_Item pointers

FL_Tree_Prefs
Tree widget's preferences

FL_Valuator
Controls a single floating-point value and provides a consistent interface to set the value, range,
and step, and insures that callbacks are done the same for every object

FL_Value_Input
Displays a numeric value

FL_Value_Output
Displays a floating point value

FL_Value_Slider
FL_Slider widget with a box displaying the current value

FL_Widget
FL_Widget is the base class for all widgets in FLTK

FL_Widget_Surface
A surface on which any FLTK widget can be drawn

FL_Widget_Tracker
This class should be used to control safe widget deletion

FL_WINDOW
This widget produces an actual window

FL_Wizard
This widget is based off the FL_Tabs widget, but instead of displaying tabs it only changes "tabs"
under program control

FL_XBM_Image
Supports loading, caching, and drawing of X Bitmap (XBM) bitmap files

FL_XColor

FL_XPM_Image
Supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency

FL_GIF_Image::GIF_FRAME

gtk_shell1_listener

Generated by Doxygen
9.1 Class List

- `gtk_surface1_listener` .............................................. 1956
- `Fl_ICO_Image::IconDirEntry`
  Windows ICONDIRENTRY structure
  1956
- `Fl_Text_Editor::Key_Binding`
  Simple linked list item associating a key/state to a function 1957
- `Fl_Terminal::Margin` ............................................. 1957
- `Fl_Preferences::Name`
  'Name' provides a simple method to create numerical or more complex procedural names for
  entries and groups on the fly ................................. 1957
- `Fl_Preferences::Node` .......................................... 1958
- `Fl_Paged_Device::page_format`
  Width, height and name of a page format .......................... 1959
- `Fl_Terminal::PartialUtf8Buf` .................................... 1960
- `Fl_Terminal::RingBuffer` ....................................... 1960
- `Fl_Preferences::RootNode` ...................................... 1961
- `Fl_Scroll::ScrollInfo`
  Structure to manage scrollbar and widget interior sizes .......... 1961
- `Fl_Terminal::Selection` ........................................ 1962
- `Fl_Tile::Size_Range` .......................................... 1962
- `Fl_Text_Display::Style_Table_Entry`
  This structure associates the color, font, and font size of a string to draw with an attribute mask
  matching attr .............................................. 1963
- `Fl_Terminal::Utf8Char` ......................................... 1963

Generated by Doxygen
Chapter 10

File Index

10.1 File List

Here is a list of all documented files with brief descriptions:

- **Enumerations.H**
  This file contains type definitions and general enumerations .................................. 1965
- **filename.H**
  File names and URI utility functions ............................................................... 1998
- **Fl.H**
  Fl static class ................................................................................................. 2000
- **Fl_Adjuster.H** .............................................................................................. 2008
- **Fl_Anim_GIF_Image.H** .................................................................................... 2009
- **fl_ask.H**
  API for common dialogs ..................................................................................... 2010
- **fl_attr.h**
  This file defines compiler-specific macros ......................................................... 2014
- **Fl_Bitmap.H** .................................................................................................. 2016
- **Fl_BMP_Image.H** ............................................................................................. 2017
- **Fl_Box.H**
  Fl_Box widget ....................................................................................................... 2018
- **Fl_Browser.H** .................................................................................................. 2018
- **Fl_Browser_.H** .................................................................................................. 2020
- **Fl_Button.H** ..................................................................................................... 2022
- **Fl_Cairo.H**
  Cairo is currently supported for the following platforms: Windows, macOS, Unix/Linux (X11 + Wayland) ................................................................. 2023
- **Fl_Cairo_Window.H**
  Fl_Cairo_Window, an FLTK window incorporating a Cairo draw callback ............... 2024
- **fl_callback_macros.H**
  This file provides macros for easy function and method callbacks with multiple type safe arguments ................................................................. 2025
- **fl_casts.H** ........................................................................................................ 2034
- **Fl_Chart.H**
  Fl_Chart widget .................................................................................................... 2034
- **Fl_Check_Browser.H** ....................................................................................... 2036
- **Fl_Check_Button.H** .......................................................................................... 2037
- **Fl_choice.H** ...................................................................................................... 2038
- **Fl_Clock.H** ....................................................................................................... 2038
- **Fl_Color_Chooser.H**
  Fl_Color_Chooser widget .................................................................................... 2039
- **fl_config.h** ....................................................................................................... 2041
- **Fl_Copy_Surface.H** ......................................................................................... 2042
- **Fl_Counter.H** ................................................................................................... 2043
<table>
<thead>
<tr>
<th>Header</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Device.H</td>
<td>2044</td>
</tr>
<tr>
<td>Declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin</td>
<td></td>
</tr>
<tr>
<td>Fl_Dial.H</td>
<td>2045</td>
</tr>
<tr>
<td>Fl_Double_Window.H</td>
<td>2046</td>
</tr>
<tr>
<td>fl_draw.H</td>
<td>2046</td>
</tr>
<tr>
<td>Utility header to pull drawing functions together</td>
<td></td>
</tr>
<tr>
<td>Fl_Export.H</td>
<td>2058</td>
</tr>
<tr>
<td>Fl_File_Browser.H</td>
<td>2058</td>
</tr>
<tr>
<td>Fl_File_Chooser.H</td>
<td>2059</td>
</tr>
<tr>
<td>Fl_File_Icon.H</td>
<td>2061</td>
</tr>
<tr>
<td>Fl_File_Input.H</td>
<td>2063</td>
</tr>
<tr>
<td>Fl_Fill_Dial.H</td>
<td>2063</td>
</tr>
<tr>
<td>Fl_Fill_Slider.H</td>
<td>2064</td>
</tr>
<tr>
<td>Fl_Flex.H</td>
<td>2064</td>
</tr>
<tr>
<td>Fl_Float_Input.H</td>
<td>2066</td>
</tr>
<tr>
<td>Fl_FormsBitmap.H</td>
<td>2066</td>
</tr>
<tr>
<td>Fl_FormsPixmap.H</td>
<td>2067</td>
</tr>
<tr>
<td>Fl_Free.H</td>
<td>2067</td>
</tr>
<tr>
<td>Fl_GIF_Image.H</td>
<td>2068</td>
</tr>
<tr>
<td>Fl_Gl_Window.H</td>
<td>2069</td>
</tr>
<tr>
<td>Fl_Graphics_Driver.H</td>
<td>2070</td>
</tr>
<tr>
<td>Fl_Grid.H</td>
<td>2075</td>
</tr>
<tr>
<td>Fl_Group.H</td>
<td></td>
</tr>
<tr>
<td>Fl_Group and Fl_End classes</td>
<td>2079</td>
</tr>
<tr>
<td>Fl_Help_Dialog.H</td>
<td>2081</td>
</tr>
<tr>
<td>Fl_Help_View.H</td>
<td>2082</td>
</tr>
<tr>
<td>Fl_Hold_Browser.H</td>
<td>2085</td>
</tr>
<tr>
<td>Fl_Hor_Fill_Slider.H</td>
<td>2085</td>
</tr>
<tr>
<td>Fl_Hor_Nice_Slider.H</td>
<td>2085</td>
</tr>
<tr>
<td>Fl_Hor_Slider.H</td>
<td>2086</td>
</tr>
<tr>
<td>Fl_Hor_Value_Slider.H</td>
<td>2086</td>
</tr>
<tr>
<td>Fl_ICO_Image.H</td>
<td>2087</td>
</tr>
<tr>
<td>Fl_Image.H</td>
<td></td>
</tr>
<tr>
<td>Fl_Image, Fl_RGB_Image classes</td>
<td>2087</td>
</tr>
<tr>
<td>Fl_Image_Surface.H</td>
<td>2090</td>
</tr>
<tr>
<td>Fl_Input.H</td>
<td>2091</td>
</tr>
<tr>
<td>Fl_Input_.H</td>
<td>2092</td>
</tr>
<tr>
<td>Fl_Input_Choice.H</td>
<td>2095</td>
</tr>
<tr>
<td>Fl_Int_Input.H</td>
<td>2097</td>
</tr>
<tr>
<td>Fl_JPEG_Image.H</td>
<td>2097</td>
</tr>
<tr>
<td>Fl_Light_Button.H</td>
<td>2097</td>
</tr>
<tr>
<td>Fl_Line_Dial.H</td>
<td>2098</td>
</tr>
<tr>
<td>Fl_Menu.H</td>
<td>2098</td>
</tr>
<tr>
<td>Fl_Menu_.H</td>
<td>2099</td>
</tr>
<tr>
<td>Fl_Menu_Bar.H</td>
<td>2100</td>
</tr>
<tr>
<td>Fl_Menu_Button.H</td>
<td>2100</td>
</tr>
<tr>
<td>Fl_Menu_Item.H</td>
<td>2101</td>
</tr>
<tr>
<td>Fl_Menu_Window.H</td>
<td>2104</td>
</tr>
<tr>
<td>fl_message.H</td>
<td>2105</td>
</tr>
<tr>
<td>Fl_Multi_Browser.H</td>
<td>2105</td>
</tr>
<tr>
<td>Fl_Multi_Label.H</td>
<td>2106</td>
</tr>
<tr>
<td>Fl_Multiline_Input.H</td>
<td>2106</td>
</tr>
<tr>
<td>Fl_Multiline_Output.H</td>
<td>2106</td>
</tr>
<tr>
<td>Fl_Native_File_Chooser.H</td>
<td>2107</td>
</tr>
<tr>
<td>Fl_Native_File_Chooser widget</td>
<td></td>
</tr>
<tr>
<td>Fl_Nice_Slider.H</td>
<td>2109</td>
</tr>
<tr>
<td>Fl_Object.H</td>
<td>2110</td>
</tr>
<tr>
<td>File</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Fl_Output.H</td>
<td>2110</td>
</tr>
<tr>
<td>Fl_Overlay_Window.H</td>
<td>2110</td>
</tr>
<tr>
<td>Fl_Pack.H</td>
<td>2111</td>
</tr>
<tr>
<td>Fl_Paged_Device.H</td>
<td>2112</td>
</tr>
<tr>
<td>Fl_Pixmap.H</td>
<td>2113</td>
</tr>
<tr>
<td>Fl_Plugin.H</td>
<td>2114</td>
</tr>
<tr>
<td>Fl_PNG_Imagen.H</td>
<td>2115</td>
</tr>
<tr>
<td>Fl_PNM_Imagen.H</td>
<td>2115</td>
</tr>
<tr>
<td>Fl_Positioner.H</td>
<td>2116</td>
</tr>
<tr>
<td>Fl_PostScript.H</td>
<td>2117</td>
</tr>
<tr>
<td>Fl_Preferences.H</td>
<td>2118</td>
</tr>
<tr>
<td>Fl_Printer.H</td>
<td>2121</td>
</tr>
<tr>
<td>Fl_Progress.H</td>
<td>2123</td>
</tr>
<tr>
<td>Fl_Radio_Button.H</td>
<td>2123</td>
</tr>
<tr>
<td>Fl_Radio_Light_Button.H</td>
<td>2124</td>
</tr>
<tr>
<td>Fl_Radio_Round_Button.H</td>
<td>2124</td>
</tr>
<tr>
<td>Fl_Rect.H</td>
<td>2125</td>
</tr>
<tr>
<td>Fl_Range_Button.H</td>
<td>2126</td>
</tr>
<tr>
<td>Fl_Return_Button.H</td>
<td>2126</td>
</tr>
<tr>
<td>Fl_RGB_Imagen.H</td>
<td>2127</td>
</tr>
<tr>
<td>Fl_Roller.H</td>
<td>2127</td>
</tr>
<tr>
<td>Fl_Round_Button.H</td>
<td>2127</td>
</tr>
<tr>
<td>Fl_Round_Clock.H</td>
<td>2128</td>
</tr>
<tr>
<td>Fl_Scheme.H</td>
<td>2128</td>
</tr>
<tr>
<td>Fl_Scheme_Choice.H</td>
<td>2129</td>
</tr>
<tr>
<td>Fl_Scroll.H</td>
<td>2129</td>
</tr>
<tr>
<td>Fl_Scrollbar.H</td>
<td>2131</td>
</tr>
<tr>
<td>Fl_Secret_Input.H</td>
<td>2131</td>
</tr>
<tr>
<td>Fl_Select_Browser.H</td>
<td>2132</td>
</tr>
<tr>
<td>Fl_Shared_Imagen.H</td>
<td>2132</td>
</tr>
<tr>
<td>Fl_Shared_Image.H</td>
<td>2132</td>
</tr>
<tr>
<td>Fl_Shortcut_Button.H</td>
<td>2135</td>
</tr>
<tr>
<td>fl_show_colormap.H</td>
<td>2135</td>
</tr>
<tr>
<td>fl_show_colormap()</td>
<td>2135</td>
</tr>
<tr>
<td>fl_show_input.H</td>
<td>2136</td>
</tr>
<tr>
<td>Fl_Simple_Counter.H</td>
<td>2136</td>
</tr>
<tr>
<td>Fl_Single_Graph.H</td>
<td>2136</td>
</tr>
<tr>
<td>Fl_Slider.H</td>
<td>2137</td>
</tr>
<tr>
<td>Fl_Spinner.H</td>
<td>2137</td>
</tr>
<tr>
<td>Fl_Scroll.H</td>
<td>2138</td>
</tr>
<tr>
<td>fl_string_functions.h</td>
<td>2138</td>
</tr>
<tr>
<td>fl_string_functions()</td>
<td>2138</td>
</tr>
<tr>
<td>Fl_SVG_File_Surface.H</td>
<td>2140</td>
</tr>
<tr>
<td>Fl_SVG_Imagen.H</td>
<td>2140</td>
</tr>
<tr>
<td>Fl_Sys_Menu_Bar.H</td>
<td>2141</td>
</tr>
<tr>
<td>Definition of class Fl_Sys_Menu_Bar</td>
<td>2142</td>
</tr>
<tr>
<td>Fl_Table.H</td>
<td>2143</td>
</tr>
<tr>
<td>Fl_Table_Row.H</td>
<td>2149</td>
</tr>
<tr>
<td>Fl_Tabs.H</td>
<td>2151</td>
</tr>
<tr>
<td>Fl_Terminal.H</td>
<td>2152</td>
</tr>
<tr>
<td>Fl_Terminal widget</td>
<td>2152</td>
</tr>
<tr>
<td>Fl_Text_Buffer.H</td>
<td>2161</td>
</tr>
<tr>
<td>Fl_Text_Display.H</td>
<td>2165</td>
</tr>
<tr>
<td>Fl_Text_Editor.H</td>
<td>2170</td>
</tr>
<tr>
<td>Fl_Tile.H</td>
<td>2171</td>
</tr>
</tbody>
</table>
Fl_Tiled_Image.H ................................................................. 2172
Fl_Timer.H ................................................................. 2173
Fl_Toggle_Button.H ......................................................... 2173
Fl_Toggle_Light_Button.H ................................................ 2174
Fl_Toggle_Round_Button.H ............................................... 2174
Fl_Tooltip.H ............................................................... 2174
Fl_Tree.H
This file contains the definitions of the Fl_Tree class ...................... 2175
Fl_Tree_Item.H
This file contains the definitions for Fl_Tree_Item .......................... 2179
Fl_Tree_Item_Array.H
This file defines a class that manages an array of Fl_Tree_Item pointers .................................................................................. 2183
Fl_Tree_Prefs.H
This file contains the definitions for Fl_Tree’s preferences ..................... 2184
fl_types.h
This file contains simple "C"-style type definitions ............................... 2190
fl_utf8.h
Header for Unicode and UTF-8 character handling ............................. 2191
Fl_Valuator.H ................................................................. 2195
Fl_Value_Input.H ............................................................. 2196
Fl_Value_Output.H ............................................................ 2197
Fl_Value_Slider.H ............................................................ 2198
Fl_Widget.H
Fl_Widget and Fl_Label classes ................................................. 2199
Fl_Widget_Surface.H .......................................................... 2204
Fl_Window.H
Fl_Window widget ........................................................................ 2205
Fl_Wizard.H ................................................................. 2208
Fl_XBM_Image.H .............................................................. 2208
Fl_XPM_Image.H .............................................................. 2209
forms.H .............................................................................. 2209
gl.h
This file defines wrapper functions for OpenGL in FLTK ...................... 2219
gl2opengl.h ................................................................. 2224
gl_draw.H ................................................................. 2224
glu.h ................................................................. 2225
glut.H ................................................................. 2225
mac.h
Mac OS X-specific symbols ....................................................... 2232
math.h .............................................................................. 2234
names.h
This file defines arrays of human readable names for FLTK symbolic constants .................................................................................. 2235
platform.H ........................................................................... 2236
platform_types.h
Definitions of platform-dependent types ........................................ 2237
wayland.H
Definitions of functions specific to the Wayland platform ....................... 2240
win32.H
Definitions of functions specific to the Windows platform ....................... 2241
x.H .............................................................................. 2243
x11.H
Definitions of functions specific to the X11 platform ............................ 2243
cgdebug.h ................................................................. 2246
fastarrow.h ................................................................. 2248
Fl.cxx
Implementation of the member functions of class Fl ................................ 2248
fl_arc.cxx
Utility functions for drawing arcs and circles ...................................... 2250

Generated by Doxygen
10.1 File List

fl_ask.cxx
Utility functions for common dialogs ................................................................. 2250
fl_boxtype.cxx
Drawing code for common box types ................................................................. 2252
fl_cmap.h ................................................................. 2254
fl_color.cxx
Color handling ................................................................. 2257
fl_compose.cxx
Utility functions to support text input ............................................................. 2257
fl_contrast.cxx
Color contrast handling ................................................................. 2258
fl_curve.cxx
Utility for drawing Bézier curves, adding the points to the current fl_begin/fl_vertex/fl_end path ................................................................. 2258
Fl_Double_Window.cxx
Fl_Double_Window implementation ................................................................. 2259
Fl_get_system_colors.cxx
System color support ................................................................. 2259
Fl_Gl_Choice.H ................................................................. 2260
Fl_Gl_Window_Driver.H ................................................................. 2261
Fl_Graphics_Driver.cxx
Implementation of class Fl_Graphics_Driver ................................................................. 2262
Fl_Grid.cxx
Implements the Fl_Grid container widget ................................................................. 2263
Fl_Image_Reader.h ................................................................. 2263
Fl_Int_Vector.H ................................................................. 2264
Fl_Message.h ................................................................. 2265
Fl_Native_File_Chooser_Kdialog.H ................................................................. 2267
Fl_Native_File_Chooser_Zenity.H ................................................................. 2268
fl_oxy.h ................................................................. 2268
Fl_Paged_Device.cxx
Implementation of class Fl_Paged_Device ................................................................. 2268
fl_rect.cxx
Drawing and clipping routines for rectangles ................................................................. 2269
Fl_Screen_Driver.H ................................................................. 2269
Fl_Sys_Menu_Bar_Driver.H ................................................................. 2273
Fl_System_Driver.H ................................................................. 2274
Fl_Timeout.cxx ................................................................. 2276
Fl_Timeout.h
Fl_Timeout handling ................................................................. 2276
fl_vertex.cxx
Portable drawing code for drawing arbitrary shapes with simple 2D transformations ................................................................. 2278
Fl_Window_Driver.H ................................................................. 2278
fl_write_png.cxx
PNG image support functions ................................................................. 2281
Fl_XColor.H ................................................................. 2283
flstring.h ................................................................. 2283
freeglut_teapot_data.h ................................................................. 2284
gtk-shell-client-protocol.h ................................................................. 2286
mediumarrow.h ................................................................. 2289
numericsort.c ................................................................. 2289
print_button.h ................................................................. 2291
print_panel.h ................................................................. 2291
slowarrow.h ................................................................. 2292
utf8_internal.h ................................................................. 2292
vsnprintf.c
Portable vsnprintf() implementation ................................................................. 2293
Xutf8.h ................................................................. 2294

Generated by Doxygen
<table>
<thead>
<tr>
<th>File Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>case.h</td>
<td>2296</td>
</tr>
<tr>
<td>dingbats_h</td>
<td>2316</td>
</tr>
<tr>
<td>spacing.h</td>
<td>2323</td>
</tr>
<tr>
<td>symbol_h</td>
<td>2345</td>
</tr>
<tr>
<td>imKStoUCS.c</td>
<td>2358</td>
</tr>
<tr>
<td>armscii_8.h</td>
<td>2362</td>
</tr>
<tr>
<td>ascii.h</td>
<td>2363</td>
</tr>
<tr>
<td>big5.h</td>
<td>2364</td>
</tr>
<tr>
<td>big5_emacs.h</td>
<td>2411</td>
</tr>
<tr>
<td>cp1133.h</td>
<td>2413</td>
</tr>
<tr>
<td>cp1251.h</td>
<td>2414</td>
</tr>
<tr>
<td>cp1255.h</td>
<td>2416</td>
</tr>
<tr>
<td>cp1256.h</td>
<td>2417</td>
</tr>
<tr>
<td>cp936ext.h</td>
<td>2419</td>
</tr>
<tr>
<td>gb2312.h</td>
<td>2491</td>
</tr>
<tr>
<td>georgian_academy.h</td>
<td>2520</td>
</tr>
<tr>
<td>georgian_ps.h</td>
<td>2521</td>
</tr>
<tr>
<td>iso8859_1.h</td>
<td>2523</td>
</tr>
<tr>
<td>iso8859_10.h</td>
<td>2523</td>
</tr>
<tr>
<td>iso8859_11.h</td>
<td>2524</td>
</tr>
<tr>
<td>iso8859_13.h</td>
<td>2525</td>
</tr>
<tr>
<td>iso8859_14.h</td>
<td>2526</td>
</tr>
<tr>
<td>iso8859_15.h</td>
<td>2528</td>
</tr>
<tr>
<td>iso8859_16.h</td>
<td>2528</td>
</tr>
<tr>
<td>iso8859_2.h</td>
<td>2530</td>
</tr>
<tr>
<td>iso8859_3.h</td>
<td>2531</td>
</tr>
<tr>
<td>iso8859_4.h</td>
<td>2532</td>
</tr>
<tr>
<td>iso8859_5.h</td>
<td>2533</td>
</tr>
<tr>
<td>iso8859_6.h</td>
<td>2534</td>
</tr>
<tr>
<td>iso8859_7.h</td>
<td>2535</td>
</tr>
<tr>
<td>iso8859_8.h</td>
<td>2536</td>
</tr>
<tr>
<td>iso8859_9.h</td>
<td>2537</td>
</tr>
<tr>
<td>iso8859_9e.h</td>
<td>2538</td>
</tr>
<tr>
<td>jisx0201.h</td>
<td>2540</td>
</tr>
<tr>
<td>jisx0208.h</td>
<td>2540</td>
</tr>
<tr>
<td>jisx0212.h</td>
<td>2568</td>
</tr>
<tr>
<td>koi8_c.h</td>
<td>2593</td>
</tr>
<tr>
<td>koi8_r.h</td>
<td>2594</td>
</tr>
<tr>
<td>koi8_u.h</td>
<td>2596</td>
</tr>
<tr>
<td>ksc5601.h</td>
<td>2598</td>
</tr>
<tr>
<td>mulelao.h</td>
<td>2632</td>
</tr>
<tr>
<td>tatar_cyr.h</td>
<td>2633</td>
</tr>
<tr>
<td>tcvn.h</td>
<td>2635</td>
</tr>
<tr>
<td>tis620.h</td>
<td>2636</td>
</tr>
<tr>
<td>ucs2be.h</td>
<td>2637</td>
</tr>
<tr>
<td>utf8.h</td>
<td>2638</td>
</tr>
<tr>
<td>viscii.h</td>
<td>2639</td>
</tr>
<tr>
<td>mk_wcwidth.c</td>
<td>2640</td>
</tr>
<tr>
<td>ucs2fontmap.c</td>
<td>2645</td>
</tr>
<tr>
<td>utf8Utils.c</td>
<td>2649</td>
</tr>
<tr>
<td>Ximint.h</td>
<td>2651</td>
</tr>
<tr>
<td>Xlibint.h</td>
<td>2651</td>
</tr>
</tbody>
</table>
Chapter 11

Topic Documentation

11.1 Callback Function Typedefs

Typedefs defined in `<FL/Fl.H>` for callback or handler functions passed as function parameters.

**Typedefs**

- `typedef void(Fl_Abort_Handler)(const char *format,...)`
  
  Signature of `set_abort` functions passed as parameters.

- `typedef int(Fl_Args_Handler)(int argc, char **argv, int &i)`
  
  Signature of `args` functions passed as parameters.

- `typedef void(Fl_Atclose_Handler)(Fl_Window *window, void *data)`
  
  Signature of `set_atclose` functions passed as parameters.

- `typedef void(Fl_Awake_Handler)(void *data)`
  
  Signature of some wakeup callback functions passed as parameters.

- `typedef void(Fl_Box_Draw_F)(int x, int y, int w, int h, Fl_Color color)`
  
  Signature of some box drawing functions passed as parameters.

- `typedef void(Fl_Clipboard_Notify_Handler)(int source, void *data)`
  
  Signature of `add_clipboard_notify` functions passed as parameters.

- `typedef int(Fl_Event_Dispatch)(int event, Fl_Window *w)`
  
  Signature of `event_dispatch` functions passed as parameters.

- `typedef int(Fl_Event_Handler)(int event)`
  
  Signature of `add_handler` functions passed as parameters.

- `typedef void(Fl_FD_Handler)(FL_SOCKET fd, void *data)`
  
  Signature of `add_fd` functions passed as parameters.

- `typedef void(Fl_Idle_Handler)(void *data)`
  
  Signature of `add_idle` callback functions passed as parameters.

- `typedef void(Fl_Timeout_Handler)(void *data)`
  
  Signature of timeout callback functions passed as parameters.
11.1.1 Detailed Description

Typedefs defined in `<FL/Fl.H>` for callback or handler functions passed as function parameters. FLTK uses callback functions as parameters for some function calls, e.g. to set up global event handlers (`Fl::add_handler()`), to add a timeout handler (`Fl::add_timeout()`), and many more. The typedefs defined in this group describe the function parameters used to set up or clear the callback functions and should also be referenced to define the callback function to handle such events in the user's code.

See also `Fl::add_handler()`, `Fl::add_timeout()`, `Fl::repeat_timeout()`, `Fl::remove_timeout()` and others

11.1.2 Typedef Documentation

11.1.2.1 `Fl_Event_Dispatch`

```c
typedef int(* Fl_Event_Dispatch) (int event, Fl_Window *w)
```

Signature of event_dispatch functions passed as parameters.

See also `Fl::event_dispatch(Fl_Event_Dispatch)`

11.1.2.2 `Fl_Timeout_Handler`

```c
typedef void(* Fl_Timeout_Handler) (void *data)
```

Signature of timeout callback functions passed as parameters. Please see `Fl::add_timeout()` for details.

11.2 Windows handling functions

Windows and standard dialogs handling declared in `<FL/Fl.H>`

Functions

- static void `Fl::default_atclose (Fl_Window *, void *)`
  
  Default callback for window widgets.

- static `Fl_Window * Fl::first_window ()`
  
  Returns the first top-level window in the list of shown() windows.

- static void `Fl::first_window (Fl_Window *)`
  
  Sets the window that is returned by `first_window()`.

- static `Fl_Window * Fl::grab ()`
  
  Returns the window that currently receives all events.

- static void `Fl::grab (Fl_Window *)`
  
  Selects the window to grab.

- static `Fl_Window * Fl::modal ()`
  
  Returns the top-most modal() window currently shown.

- static `Fl_Window * Fl::next_window (const Fl_Window *)`
  
  Returns the next top-level window in the list of shown() windows.

- static void `Fl::set_abort (Fl_Abort_Handler f)`
  
  For back compatibility, sets the void `Fl::fatal` handler callback.

- static void `Fl::set_atclose (Fl_Atclose_Handler f)`
  
  For back compatibility, sets the `Fl::atclose` handler callback.

Variables

- static void(* `Fl::atclose (Fl_Window *, void *)`)
  
  Back compatibility: default window callback handler.
11.2 Windows handling functions

11.2.1 Detailed Description
Windows and standard dialogs handling declared in `<FL/FL.H>`

11.2.2 Function Documentation

11.2.2.1 default_atclose()

```c
void Fl::default_atclose ( 
    Fl_Window * window, 
    void * v ) [static]
```

Default callback for window widgets.
It hides the window and then calls the default widget callback.

11.2.2.2 first_window [1/2]

```c
Fl_Window * Fl::first_window ( ) [static]
```

Returns the first top-level window in the list of shown() windows.
If a modal() window is shown this is the top-most modal window, otherwise it is the most recent window to get an event.

11.2.2.3 first_window [2/2]

```c
void Fl::first_window ( 
    Fl_Window * window ) [static]
```

Sets the window that is returned by first_window().
The window is removed from wherever it is in the list and inserted at the top. This is not done if Fl::modal() is on or if the window is not shown(). Because the first window is used to set the "parent" of modal windows, this is often useful.

11.2.2.4 grab [1/2]

```c
static Fl_Window * Fl::grab ( ) [inline], [static]
```

Returns the window that currently receives all events.

Optional returns
- The window that currently receives all events, or NULL if event grabbing is currently OFF.

11.2.2.5 grab [2/2]

```c
void Fl::grab ( 
    Fl_Window * win ) [static]
```

Selects the window to grab.
This is used when pop-up menu systems are active.
Send all events to the passed window no matter where the pointer or focus is (including in other programs). The window does not have to be shown(). This lets the handle() method of a "dummy" window override all event handling and allows you to map and unmap a complex set of windows (under both X and Windows some window must be mapped because the system interface needs a window id).
If grab() is on it will also affect show() of windows by doing system-specific operations (on X it turns on override-redirect). These are designed to make menus popup reliably and faster on the system.
To turn off grabbing do Fl::grab(0).
Be careful that your program does not enter an infinite loop while grab() is on. On X this will lock up your screen!
To avoid this potential lockup, all newer operating systems seem to limit mouse pointer grabbing to the time during which a mouse button is held down. Some OS's may not support grabbing at all.

11.2.2.6 modal()

```c
static Fl_Window * Fl::modal ( ) [inline], [static]
```

Returns the top-most modal() window currently shown.

Generated by Doxygen
This is the most recently shown() window with modal() true, or NULL if there are no modal() windows shown(). The modal() window has its handle() method called for all events, and no other windows will have handle() called (grab() overrides this).

### 11.2.2.7 next_window()

```c
Fl_Window * Fl::next_window (const Fl_Window * window ) [static]
```

Returns the next top-level window in the list of shown() windows. You can use this call to iterate through all the windows that are shown().

Parameters

- **window** must be shown and not NULL

### 11.2.2.8 set_atclose()

```c
static void Fl::set_atclose (Fl_Atclose_Handler f) [inline], [static]
```

For back compatibility, sets the Fl::atclose handler callback. You can now simply change the callback for the window instead.

See also

- `Fl_Window::callback(Fl_Callback*)`

### 11.2.3 Variable Documentation

#### 11.2.3.1 atclose

```c
void(* Fl::atclose)(Fl_Window *, void *)=default_atclose [static], [default]
```

Back compatibility: default window callback handler.

See also

- `Fl::set_atclose()`

### 11.3 Events handling functions

Fl class events handling API declared in `<FL/FL.H>`

**Functions**

- static void Fl::add_handler (Fl_Event_Handler h)
  
  Install a function to parse unrecognized events.

- static void Fl::add_system_handler (Fl_System_Handler h, void *data)
  
  Install a function to intercept system events.

- static Fl_Widget * Fl::belowmouse ()
  
  Gets the widget that is below the mouse.

- static void Fl::belowmouse (Fl_Widget *)
  
  Sets the widget that is below the mouse.

- static Fl_Callback_Reason Fl::callback_reason ()
  
  Give the reason for calling a callback.

- static int Fl::compose (int &del)
  
  Any text editing widget should call this for each FL_KEYBOARD event.

- static void Fl::compose_reset ()
  
  If the user moves the cursor, be sure to call Fl::compose_reset().
11.3 Events handling functions

- static void Fl::disable_im()
  
  Disables the system input methods facilities.

- static void Fl::enable_im()
  
  Enables the system input methods facilities.

- static int Fl::event()
  
  Returns the last event that was processed.

- static int Fl::event_alt()
  
  Returns non-zero if the Alt key is pressed.

- static int Fl::event_button()
  
  Gets which particular mouse button caused the current event.

- static int Fl::event_button1()
  
  Returns non-zero if mouse button 1 is currently held down.

- static int Fl::event_button2()
  
  Returns non-zero if button 2 is currently held down.

- static int Fl::event_button3()
  
  Returns non-zero if button 3 is currently held down.

- static int Fl::event_buttons()
  
  Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.

- static int Fl::event_clicks()
  
  Returns non-zero if we had a double click event.

- static void Fl::event_clicks(int i)
  
  Manually sets the number returned by Fl::event_clicks().

- static void * Fl::event_clipboard()
  
  During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.

- static const char * Fl::event_clipboard_type()
  
  Returns the type of the pasted data during an FL_PASTE event.

- static int Fl::event_command()
  
  Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.

- static int Fl::event_ctrl()
  
  Returns non-zero if the Control key is pressed.

- static Fl::Event_Dispatch Fl::event_dispatch()
  
  Return the current event dispatch function.

- static void Fl::event_dispatch(Fl::Event_Dispatch d)
  
  Set a new event dispatch function.

- static int Fl::event_dx()
  
  Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.

- static int Fl::event_dy()
  
  Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.

- static int Fl::event_inside(const Fl_Widget *)
  
  Returns whether or not the mouse event is inside a given child widget.

- static int Fl::event_inside(int, int, int, int)
  
  Returns whether or not the mouse event is inside the given rectangle.

- static int Fl::event_is_click()
  
  Returns non-zero if the mouse has not moved far enough and not enough time has passed since the last FL_PUSH or FL_KEYBOARD event for it to be considered a "drag" rather than a "click".

- static void Fl::event_is_click(int i)
  
  Clears the value returned by Fl::event_is_click().

- static int Fl::event_key()
  
  Gets which key on the keyboard was last pushed.

- static int Fl::event_key(int key)
  
  Returns true if the given key was held down (or pressed) during the last event.
• static int Fl::event_length ()
  Returns the length of the text in Fl::event_text().

• static int Fl::event_original_key ()
  Returns the keycode of the last key event, regardless of the NumLock state.

• static int Fl::event_shift ()
  Returns non-zero if the Shift key is pressed.

• static int Fl::event_state ()
  Returns the keyboard and mouse button states of the last event.

• static int Fl::event_state (int mask)
  Returns non-zero if any of the passed event state bits are turned on.

• static const char * Fl::event_text ()
  Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.

• static int Fl::event_x ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.

• static int Fl::event_x_root ()
  Returns the mouse position on the screen of the event.

• static int Fl::event_y ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.

• static int Fl::event_y_root ()
  Returns the mouse position on the screen of the event.

• static Fl_Widget * Fl::focus ()
  Gets the current Fl::focus() widget.

• static void Fl::focus (Fl_Widget *)
  Sets the widget that will receive FL_KEYBOARD events.

• static int Fl::get_key (int key)
  Returns true if the given key is held down now.

• static void Fl::get_mouse (int &, int &)
  Return where the mouse is on the screen by doing a round-trip query to the server.

• static int Fl::handle (int, Fl_Window *)
  Handle events from the window system.

• static int Fl::handle_ (int, Fl_Window *)
  Handle events from the window system.

• static Fl_Widget * Fl::pushed ()
  Gets the widget that is being pushed.

• static void Fl::pushed (Fl_Widget *)
  Sets the widget that is being pushed.

• static void Fl::remove_handler (Fl_Event_Handler h)
  Removes a previously added event handler.

• static void Fl::remove_system_handler (Fl_System_Handler h)
  Removes a previously added system event handler.

• static int Fl::test_shortcut (Fl_Shortcut)
  Tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value (described in Fl_Button).

Variables

• const char *const fl_callback_reason_names []
  This is an array of callback reason names you can use to convert font numbers into names.

• const char *const fl_eventnames []
  This is an array of event names you can use to convert event numbers into names.

• const char *const fl_fontnames []
  This is an array of font names you can use to convert font numbers into names.
11.3 Events handling functions

11.3.1 Detailed Description

FL class events handling API declared in `<FL/FL.H>`

11.3.2 Function Documentation

11.3.2.1 `add_handler()`

```c
void Fl::add_handler ( 
    Fl_Event_Handler ha ) [static]
```

Install a function to parse unrecognized events. If FLTK cannot figure out what to do with an event, it calls each of these functions (most recent first) until one of them returns non-zero. If none of them returns non-zero then the event is ignored. Events that cause this to be called are:

- FL_SHORTCUT events that are not recognized by any widget. This lets you provide global shortcut keys.
- FL_SCREEN_CONFIGURATION_CHANGED events. Under X11, this event requires the libXrandr.so shared library to be loadable at run-time and the X server to implement the RandR extension.
- FL_ZOOM_EVENT events.
- System events that FLTK does not recognize. See `fl_xevent`.
- Some other events when the widget FLTK selected returns zero from its `handle()` method. Exactly which ones may change in future versions, however.

See also

- `Fl::remove_handler(Fl_Event_Handler)`
- `Fl::event_dispatch(Fl_Event_Dispatch d)`
- `Fl::handle(int, Fl_Window*)`

11.3.2.2 `add_system_handler()`

```c
void Fl::add_system_handler ( 
    Fl_System_Handler ha, 
    void * data ) [static]
```

Install a function to intercept system events. FLTK calls each of these functions as soon as a new system event is received. The processing will stop at the first function to return non-zero. If all functions return zero then the event is passed on for normal handling by FLTK. Each function will be called with a pointer to the system event as the first argument and `data` as the second argument. The system event pointer will always be void *, but will point to different objects depending on the platform:

- X11: XEvent
- Windows: MSG
- OS X: NSEvent

Parameters

<table>
<thead>
<tr>
<th>ha</th>
<th>The event handler function to register</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>User data to include on each call</td>
</tr>
</tbody>
</table>

See also

- `Fl::remove_system_handler(Fl_System_Handler)`
11.3.2.3 **belowmouse()** [1/2]

```cpp
static Fl_Widget * Fl::belowmouse ( ) [inline], [static]
```

Gets the widget that is below the mouse.

See also

```cpp
belowmouse(Fl_Widget*)
```

11.3.2.4 **belowmouse()** [2/2]

```cpp
void Fl::belowmouse ( Fl_Widget * o ) [static]
```

Sets the widget that is below the mouse.

This is for highlighting buttons. It is not used to send FL_PUSH or FL_MOVE directly, for several obscure reasons, but those events typically go to this widget. This is also the first widget tried for FL_SHORTCUT events.

If you change the belowmouse widget, the previous one and all parents (that don't contain the new widget) are sent FL_LEAVE events. Changing this does not send FL_ENTER to this or any widget, because sending FL_ENTER is supposed to test if the widget wants the mouse (by it returning non-zero from handle()).

11.3.2.5 **callback_reason()**

```cpp
Fl_Callback_Reason Fl::callback_reason ( ) [static]
```

Give the reason for calling a callback.

Returns

the reason for the current callback

See also

```cpp
Fl_Widget::when(), Fl_Widget::do_callback(), Fl_Widget::callback()
```

11.3.2.6 **compose()**

```cpp
int Fl::compose ( int & del ) [static]
```

Any text editing widget should call this for each FL_KEYBOARD event.

Use of this function is very simple.

If **true** is returned, then it has modified the `Fl::event_text()` and `Fl::event_length()` to a set of bytes to insert (it may be of zero length!). It will also set the "del" parameter to the number of bytes to the left of the cursor to delete, this is used to delete the results of the previous call to `Fl::compose()`.

If **false** is returned, the keys should be treated as function keys, and del is set to zero. You could insert the text anyways, if you don't know what else to do.

Text editing widgets can preferentially call `fl_set_spot()` to indicate the window coordinates of the bottom of the current insertion point and the line height. This way, auxiliary windows that help choosing among alternative characters with some text input methods appear just below or above the insertion point. If widgets don't do that, such auxiliary windows appear at the widget's bottom.

On some platforms, text input can involve marked text, that is, temporary text replaced by other text during the input process. This occurs, e.g., under Wayland or macOS when using dead keys or when entering CJK characters. Text editing widgets should preferentially signal marked text, usually underlining it..Widgets can use `int Fl::compose_state after having called Fl::compose(int&)` to obtain the length in bytes of marked text that always finishes at the current insertion point. Widgets should also call void `fl_reset_spot()` when processing FL_UNFOCUS events. The `Fl_Input` and `Fl_Text_Editor` widgets underline marked text. If none of this is done by a user-defined text editing widget, text input will work, but will not signal to the user what text is marked.

Finally, text editing widgets should call `set_flag(MAC_USE_ACCENTS_MENU);` in their constructor if they want to use, on the macOS platform, the feature introduced with Mac OS 10.7 "Lion" where pressing and holding certain keys on the keyboard opens a diacritic marks popup window.
11.3 Events handling functions

Note

For compatibility with FLTK 1.3, text editing widgets can call `Fl::insertion_point_location(int x, int y, int height)` and `Fl::reset_marked_text()` only under the macOS platform to indicate/reset the coordinates of the current insertion point. This is deprecated in version 1.4 because redundant with the platform-independent `fl_set_spot()` and `fl_reset_spot()` functions.

11.3.2.7 compose_reset()

```cpp
void Fl::compose_reset ( ) [static]
```

If the user moves the cursor, be sure to call `Fl::compose_reset()`. The next call to `Fl::compose()` will start out in an initial state. In particular it will not set "del" to non-zero. This call is very fast so it is ok to call it many times and in many places.

11.3.2.8 disable_im()

```cpp
void Fl::disable_im ( ) [static]
```

Disables the system input methods facilities.

See also

```cpp
enable_im()
```

11.3.2.9 enable_im()

```cpp
void Fl::enable_im ( ) [static]
```

Enables the system input methods facilities. This is the default.

See also

```cpp
disable_im()
```

11.3.2.10 event()

```cpp
static int Fl::event ( ) [inline], [static]
```

Returns the last event that was processed. This can be used to determine if a callback is being done in response to a keypress, mouse click, etc.

11.3.2.11 event_button()

```cpp
static int Fl::event_button ( ) [inline], [static]
```

Gets which particular mouse button caused the current event. This returns garbage if the most recent event was not a FL_PUSH or FL_RELEASE event.

Return values

<table>
<thead>
<tr>
<th>FL_LEFT_MOUSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_MIDDLE_MOUSE</td>
</tr>
<tr>
<td>FL_RIGHT_MOUSE</td>
</tr>
</tbody>
</table>

See also

```cpp
Fl::event_buttons()
```

11.3.2.12 event_button1()

```cpp
static int Fl::event_button1 ( ) [inline], [static]
```
Returns non-zero if mouse button 1 is currently held down.
For more details, see Fl::event_buttons().

11.3.2.13  event_button2()

static int Fl::event_button2 ( ) [inline], [static]
Returns non-zero if button 2 is currently held down.
For more details, see Fl::event_buttons().

11.3.2.14  event_button3()

static int Fl::event_button3 ( ) [inline], [static]
Returns non-zero if button 3 is currently held down.
For more details, see Fl::event_buttons().

11.3.2.15  event_buttons()

static int Fl::event_buttons ( ) [inline], [static]
Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.
This function returns the button state at the time of the event. During an FL_RELEASE event, the state of the
released button will be 0. To find out, which button caused an FL_RELEASE event, you can use Fl::event_button()
instead.

Returns
a bit mask value like \{ FL_BUTTON1 | FL_BUTTON2 | FL_BUTTON3 \}

11.3.2.16  event_clicks() [1/2]

static int Fl::event_clicks ( ) [inline], [static]
Returns non-zero if we had a double click event.

Return values

| Non-zero | if the most recent FL_PUSH or FL_KEYBOARD was a "double click". |
| N-1      | for N clicks. A double click is counted if the same button is pressed again while event_is_click() is true. |

11.3.2.17  event_clicks() [2/2]

static void Fl::event_clicks (int i ) [inline], [static]
Manually sets the number returned by Fl::event_clicks().
This can be used to set it to zero so that later code does not think an item was double-clicked.

Parameters

| in     | i | corresponds to no double-click if 0, i+1 mouse clicks otherwise |

See also

int event_clicks()

11.3.2.18  event_clipboard()

static void * Fl::event_clipboard ( ) [inline], [static]
During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.
The returned data is an Fl_RGB_Image * when the result of Fl::event_clipboard_type() is Fl::clipboard_image.
11.3 Events handling functions

11.3.2.19  event_clipboard_type()

static const char * Fl::event_clipboard_type () [inline], [static]
Returns the type of the pasted data during an FL_PASTE event.
This type can be Fl::clipboard_plain_text or Fl::clipboard_image.

11.3.2.20  event_dispatch()

void Fl::event_dispatch (      
   Fl_Event_Dispatch d ) [static]
Set a new event dispatch function.
The event dispatch function is called after native events are converted to FLTK events, but before they are handled by FLTK. If the dispatch function Fl_Event_Dispatch d is set, it is up to the dispatch function to call Fl::handle_(int, Fl_Window*) or to ignore the event.
The dispatch function itself must return 0 if it ignored the event, or non-zero if it used the event. If you call Fl::handle_(), then this will return the correct value.
The event dispatch can be used to handle exceptions in FLTK events and callbacks before they reach the native event handler:

```c
int myHandler(int e, Fl_Window *w) {  
   try {  
      return Fl::handle_(e, w);  
   } catch (...) {  
      ...  
   }  
}

main() {  
   Fl::event_dispatch(myHandler);  
   Fl::run();  
}
```

Parameters

| d | new dispatch function, or NULL |

See also

Fl::add_handler(Fl_Event_Handler)
Fl::handle(int, Fl_Window*)
Fl::handle_(int, Fl_Window*)

11.3.2.21  event_dx()

static int Fl::event_dx ( ) [inline], [static]
Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
Right is positive.

11.3.2.22  event_dy()

static int Fl::event_dy ( ) [inline], [static]
Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
Down is positive.

11.3.2.23  event_inside() [1/2]

int Fl::event_inside (      
   const Fl_Widget * o ) [static]
Returns whether or not the mouse event is inside a given child widget.
Returns non-zero if the current Fl::event_x() and Fl::event_y() put it inside the given child widget's bounding box.
This method can only be used to check whether the mouse event is inside a child widget of the window that handles the event, and there must not be an intermediate subwindow (i.e. the widget must not be inside a subwindow of the current window). However, it is valid if the widget is inside a nested Fl_Group.
You must not use it with the window itself as the \o argument in a window's handle() method.

**Note**

The mentioned restrictions are necessary, because this method does not transform coordinates of child widgets, and thus the given widget \o must be within the same window that is handling the current event. Otherwise the results are undefined.

You should always call this rather than doing your own comparison so you are consistent about edge effects.

**See also**

Fl::event_inside(int, int, int)

**Parameters**

- in \o child widget to be tested

**Returns**

non-zero, if mouse event is inside the widget

### 11.3.2.24 event_inside() [2/2]

```c
int Fl::event_inside ( int xx,
                  int yy,
                  int ww,
                  int hh ) [static]
```

Returns whether or not the mouse event is inside the given rectangle. Returns non-zero if the current Fl::event_x() and Fl::event_y() put it inside the given arbitrary bounding box. You should always call this rather than doing your own comparison so you are consistent about edge effects. To find out, whether the event is inside a child widget of the current window, you can use Fl::event_inside(const Fl_Widget *).

**Parameters**

- in xx,yy,ww,hh bounding box

**Returns**

non-zero, if mouse event is inside

### 11.3.2.25 event_is_click() [1/2]

```c
static int Fl::event_is_click () [inline], [static]
```

Returns non-zero if the mouse has not moved far enough and not enough time has passed since the last FL_PUSH or FL_KEYBOARD event for it to be considered a "drag" rather than a "click". You can test this on FL_DRAG, FL_RELEASE, and FL_MOVE events.

### 11.3.2.26 event_is_click() [2/2]

```c
static void Fl::event_is_click ( int i ) [inline], [static]
```

Clears the value returned by Fl::event_is_click(). Useful to prevent the next click from being counted as a double-click or to make a popup menu pick an item with a single click. Don't pass non-zero to this.
11.3 Events handling functions

11.3.2.27  **event_key()** [1/2]

```c
static int Fl::event_key ( ) [inline], [static]
```

Gets which key on the keyboard was last pushed. The returned integer ‘key code’ is not necessarily a text equivalent for the keystroke. For instance: if someone presses ‘5’ on the numeric keypad with numlock on, Fl::event_key() may return the ‘key code’ for this key, and NOT the character ‘5’. To always get the ‘5’, use Fl::event_text() instead.

Returns

an integer ‘key code’, or 0 if the last event was not a key press or release.

See also

int event_key(int), event_text(), compose(int&).

11.3.2.28  **event_key()** [2/2]

```c
int Fl::event_key ( int key ) [static]
```

Returns true if the given **key** was held down (or pressed) **during** the last event. This is constant until the next event is read from the server.

Fl::get_key(int) returns true if the given key is held down **now**. Under X this requires a round-trip to the server and is **much** slower than Fl::event_key(int).

Keys are identified by the **unshifted** values. FLTK defines a set of symbols that should work on most modern machines for every key on the keyboard:

- All keys on the main keyboard producing a printable ASCII character use the value of that ASCII character (as though shift, ctrl, and caps lock were not on). The space bar is 32.
- All keys on the numeric keypad producing a printable ASCII character use the value of that ASCII character plus FL_KP (e.g., FL_KP + '4', FL_KP + '/'). The highest possible value is FL_KP_Last so you can range-check to see if something is on the keypad.
- All numbered function keys use the number on the function key plus FL_F. The highest possible number is FL_F_Last, so you can range-check a value.
- Buttons on the mouse are considered keys, and use the button number (where the left button is 1) plus FL_Button.
- All other keys on the keypad have a symbol: FL_Escape, FL_BackSpace, FL_Tab, FL_Enter, FL_Print, FL←Scroll_Lock, FL_Pause, FL_Insert, FL_Home, FL_Page_Up, FL_Delete, FL_End, FL_Page_Down, FL_Left, FL_Up, FL_Right, FL_Down, FL_Iso_Key, FL_Shift_L, FL_Shift_R, FL_Control_L, FL_Control_R, FL_Caps←Lock, FL_Alt_L, FL_Alt_R, FL_Meta_L, FL_Meta_R, FL_Menu, FL_Num_Lock, FL_KP_Enter. Be careful not to confuse these with the very similar, but all-caps, symbols used by Fl::event_state().

On X Fl::get_key(FL_Button+n) does not work.
On Windows Fl::get_key(FL_KP_Enter) and Fl::event_key(FL_KP_Enter) do not work.

11.3.2.29  **event_length()**

```c
static int Fl::event_length ( ) [inline], [static]
```

Returns the length of the text in Fl::event_text(). There will always be a nul at this position in the text. However there may be a nul before that if the keystroke translates to a nul character or you paste a nul character.

11.3.2.30  **event_original_key()**

```c
static int Fl::event_original_key ( ) [inline], [static]
```

Returns the keycode of the last key event, regardless of the NumLock state. If NumLock is deactivated, FLTK translates events from the numeric keypad into the corresponding arrow key events. **event_key()** returns the translated key code, whereas **event_original_key()** returns the keycode before NumLock translation.

Generated by Doxygen
11.3.2.31  event_state() [1/2]

static int Fl::event_state ( ) [inline], [static]
Returns the keyboard and mouse button states of the last event.
This is a bitfield of what shift states were on and what mouse buttons were held down during the most recent event.
The legal event state bits are:

- FL_SHIFT
- FL_CAPS_LOCK
- FL_CTRL
- FL_ALT
- FL_NUM_LOCK
- FL_META
- FL_SCROLL_LOCK
- FL_BUTTON1
- FL_BUTTON2
- FL_BUTTON3

X servers do not agree on shift states, and FL_NUM_LOCK, FL_META, and FL_SCROLL_LOCK may not work.
The values were selected to match the XFree86 server on Linux. In addition there is a bug in the way X works so
that the shift state is not correctly reported until the first event after the shift key is pressed or released.

11.3.2.32  event_state() [2/2]

static int Fl::event_state (int mask) [inline], [static]
Returns non-zero if any of the passed event state bits are turned on.
Use mask to pass the event states you're interested in. The legal event state bits are defined in Fl::event_state().

11.3.2.33  event_text()

static const char * Fl::event_text ( ) [inline], [static]
Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.
This can be used in response to FL_KEYUP, FL_KEYDOWN, FL_PASTE, and FL_DND_RELEASE.
When responding to FL_KEYUP/FL_KEYDOWN, use this function instead of Fl::event_key() to get the text equivalent of keystrokes suitable for inserting into strings and text widgets.
The returned string is guaranteed to be NULL terminated. However, see Fl::event_length() for the actual length of
the string, in case the string itself contains NULLs that are part of the text data.

Returns
A NULL terminated text string equivalent of the last keystroke.

11.3.2.34  event_x_root()

static int Fl::event_x_root ( ) [inline], [static]
Returns the mouse position on the screen of the event.
To find the absolute position of an Fl_Window on the screen, use the difference between event_x_root(),event_y_root() and
event_x(),event_y().

11.3.2.35  event_y_root()

static int Fl::event_y_root ( ) [inline], [static]
Returns the mouse position on the screen of the event.
To find the absolute position of an Fl_Window on the screen, use the difference between event_x_root(),event_y_root() and
event_x(),event_y().
11.3.2.36 focus() [1/2]

static Fl_Widget * Fl::focus ( ) [inline], [static]
Gets the current Fl::focus() widget.
See also

Fl::focus(Fl_Widget *)

11.3.2.37 focus() [2/2]

void Fl::focus ( Fl_Widget * o ) [static]
Sets the widget that will receive FL_KEYBOARD events.
Use this function inside the handle(int) member function of a widget of yours to give focus to the widget, for example when it receives the FL_FOCUS or the FL_PUSH event. Otherwise, use Fl_Widget::take_focus() to give focus to a widget;
If you change Fl::focus(), the previous widget and all parents (that don't contain the new widget) are sent FL←UNFOCUS events. Changing the focus does not send FL_FOCUS to this or any widget, because sending FL←FOCUS is supposed to test if the widget wants the focus (by it returning non-zero from handle()).
Widgets can set the NEEDS_KEYBOARD flag to indicate that a keyboard is essential for the widget to function. Touchscreen devices will be sent a request to show an on-screen keyboard if no hardware keyboard is connected.
See also

Fl_Widget::take_focus()
Fl_Widget::needs_keyboard() const
Fl_Widget::needs_keyboard(bool)

11.3.2.38 get_key()

int Fl::get_key ( int key ) [static]
Returns true if the given key is held down now.
Under X this requires a round-trip to the server and is much slower than Fl::event_key(int).
See also

event_key(int)

11.3.2.39 get_mouse()

void Fl::get_mouse ( int & x, int & y ) [static]
Return where the mouse is on the screen by doing a round-trip query to the server.
you should use Fl::event_x_root() and Fl::event_y_root() if possible, but this is necessary if you are not sure if a mouse event has been processed recently (such as to position your first window). If the display is not open, this will open it.

11.3.2.40 handle()

int Fl::handle ( int e, Fl_Window * window ) [static]
Handle events from the window system.
This is called from the native event dispatch after native events have been converted to FLTK notation. This function calls Fl::handle_(int, Fl_Window+) unless the user sets a dispatch function. If a user dispatch function is set, the user must make sure that Fl::handle_(i) is called, or the event will be ignored.
Parameters

<table>
<thead>
<tr>
<th>$e$</th>
<th>the event type ($FL::event_number()$ is not yet set)</th>
</tr>
</thead>
<tbody>
<tr>
<td>window</td>
<td>the window that caused this event</td>
</tr>
</tbody>
</table>

Returns

0 if the event was not handled

See also

$FL::add_handler(FL\_EventHandler)$

$FL::event\_dispatch(FL\_Event\_Dispatch)$

11.3.2.41 handle()

```c
int Fl::handle_ (  
    int $e$,  
    FL_Window * window ) [static]
```
Handle events from the window system.
This function is called from the native event dispatch, unless the user sets another dispatch function. In that case, the user dispatch function must decide when to call $FL::handle_\_\_ (int, FL\_Window*)$

Callbacks can set $FL\_REASON\_CLOSED$ and $FL\_REASON\_CANCELLED$.

Parameters

<table>
<thead>
<tr>
<th>$e$</th>
<th>the event type ($FL::event_number()$ is not yet set)</th>
</tr>
</thead>
<tbody>
<tr>
<td>window</td>
<td>the window that caused this event</td>
</tr>
</tbody>
</table>

Returns

0 if the event was not handled

See also

$FL::event\_dispatch(FL\_Event\_Dispatch)$

11.3.2.42 pushed() [1/2]

```c
static FL_Widget * Fl::pushed ( ) [inline], [static]
```
Gets the widget that is being pushed.

See also

void pushed(FL_Widget*)

11.3.2.43 pushed() [2/2]

```c
void Fl::pushed (  
    FL_Widget * o ) [static]
```
Sets the widget that is being pushed.

$FL\_DRAG$ or $FL\_RELEASE$ (and any more $FL\_PUSH$) events will be sent to this widget.
If you change the pushed widget, the previous one and all parents (that don’t contain the new widget) are sent $FL\_RELEASE$ events. Changing this does not send $FL\_PUSH$ to this or any widget, because sending $FL\_PUSH$ is supposed to test if the widget wants the mouse (by it returning non-zero from $handle()$).
11.3 Events handling functions

### 11.3.2.44 remove_handler()

```c
void Fl::remove_handler ( 
   Fl_Event_Handler ha ) [static]
```

Removes a previously added event handler.

See also

`Fl::handle(int, Fl_Winodw*)`

### 11.3.2.45 remove_system_handler()

```c
void Fl::remove_system_handler ( 
   Fl_System_Handler ha ) [static]
```

Removes a previously added system event handler.

**Parameters**

- `ha` The event handler function to remove

See also

`Fl::add_system_handler(Fl_System_Handler)`

### 11.3.2.46 test_shortcut()

```c
int Fl::test_shortcut ( 
   Fl_Shortcut shortcut ) [static]
```

Tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value (described in Fl_Button).

Not to be confused with Fl_Widget::test_shortcut().

**Returns**

non-zero if there is a match.

### 11.3.3 Variable Documentation

#### 11.3.3.1 fl_callback_reason_names

```c
const char * const fl_callback_reason_names[]
```

**Initial value:**

```c
= [ 
   "FL_REASON_UNKNOWN", 
   "FL_REASON_SELECTED", 
   "FL_REASON_DESELECTED", 
   "FL_REASON_RESELECTED", 
   "FL_REASON_OPENED", 
   "FL_REASON_CLOSED", 
   "FL_REASON_DRAGGED", 
   "FL_REASON_CANCELLED", 
   "FL_REASON_CHANGED", 
   "FL_REASON_GOT_FOCUS", 
   "FL_REASON_LOST_FOCUS", 
   "FL_REASON_RELEASED", 
   "FL_REASON_ENTER_KEY", 
   NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, 
   NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, 
   "FL_REASON_USER", "FL_REASON_USER+1", "FL_REASON_USER+2", "FL_REASON_USER+3", 
]}
```

This is an array of callback reason names you can use to convert font numbers into names.

The array gets defined inline wherever your '#include <FL/names.h>' appears.
11.3.3.2 fl_eventnames

const char* const fl_eventnames[]

This is an array of event names you can use to convert event numbers into names. The array gets defined inline wherever your `#include <FL/names.h>` appears.

Example:

```c
#include <FL/names.h> // array will be defined here
int MyClass::handle(int e) {
    printf("Event was %s (%d)\n", fl_eventnames[e], e);
    // ..resulting output might be e.g. "Event was FL_PUSH (1)"..
    [...]}
```

11.3.3.3 fl_fontnames

const char* const fl_fontnames[]

Initial value:

```
{ "FL_HELVETICA", 
  "FL_HELVETICA_BOLD", 
  "FL_HELVETICA_ITALIC", 
  "FL_HELVETICA_BOLD_ITALIC", 
  "FL_COURIER", 
  "FL_COURIER_BOLD", 
  "FL_COURIER_ITALIC", 
  "FL_COURIER_BOLD_ITALIC", 
  "FL_TIMES", 
  "FL_TIMES_BOLD", 
  "FL_TIMES_ITALIC", 
  "FL_TIMES_BOLD_ITALIC", 
  "FL_SYMBOL", 
  "FL_SCREEN", 
  "FL_SCREEN_BOLD", 
  "FL_ZAPF_DINGBATS", }
```

This is an array of font names you can use to convert font numbers into names. The array gets defined inline wherever your `#include <FL/names.h>` appears.

Example:

```c
#include <FL/names.h> // array will be defined here
int MyClass::my_callback(Fl_Widget *w, void*) {
    int fnum = w->labelfont();
    // Resulting output might be e.g. "Label’s font is FL_HELVETICA (0)"
    printf("Label’s font is %s (%d)\n", fl_fontnames[fnum], fnum);
    // ..resulting output might be e.g. "Label’s font is FL_HELVETICA (0)"..
    [...]}
```

11.4 Selection & Clipboard functions

FLTK global copy/cut/paste functions declared in `<FL/Fl.H>`

Functions

- static void Fl::add_clipboard_notify (Fl_Clipboard_Notify_Handler h, void *data=0)
  
  FLTK will call the registered callback whenever there is a change to the selection buffer or the clipboard.

- static int Fl::clipboard_contains (const char *type)
  
  Returns non 0 if the clipboard contains data matching `type`.

- static void Fl::copy (const char *stuff, int len, int destination=0, const char *type=Fl::clipboard_plain_text)
  
  Copies the data pointed to by `stuff` to the selection buffer (`destination` is 0), the clipboard (`destination` is 1), or both (`destination` is 2).

- static int Fl::dnd ()
  
  Initiate a Drag And Drop operation.

- static void Fl::paste (Fl_Widget &receiver)
  
  Backward compatibility only.

- static void Fl::paste (Fl_Widget &receiver, int source, const char *type=Fl::clipboard_plain_text)
  
  Pastes the data from the selection buffer (`source` is 0) or the clipboard (`source` is 1) into `receiver`.

- static void Fl::remove_clipboard_notify (Fl_Clipboard_Notify_Handler h)
Stop calling the specified callback when there are changes to the selection buffer or the clipboard.

- `static void Fl::selection (Fl_Widget &owner, const char *, int len)`
  Changes the current selection.

- `static Fl_Widget * Fl::selection_owner ()`
  back-compatibility only: Gets the widget owning the current selection

- `static void Fl::selection_owner(Fl_Widget *)`
  Back-compatibility only: The single-argument call can be used to move the selection to another widget or to set the owner to NULL, without changing the actual text of the selection.

- `static int Fl::selection_to_clipboard ()`
  Returns the current selection_to_clipboard mode.

- `static void Fl::selection_to_clipboard (int mode)`
  Copies selections on X11 directly to the clipboard if enabled.

**Variables**

- `static char const *const Fl::clipboard_image = "image"`
  Denotes image data.

- `static char const *const Fl::clipboard_plain_text = "text/plain"`
  Denotes plain textual data.

11.4.1 Detailed Description

FLTK global copy/cut/paste functions declared in `<FL/Fl.H>`

11.4.2 Function Documentation

11.4.2.1 add_clipboard_notify()

```c

void Fl::add_clipboard_notify (  
    Fl_Clipboard_Notify_Handler h,  
    void * data = 0 )  
```

FLTK will call the registered callback whenever there is a change to the selection buffer or the clipboard. The source argument indicates which of the two has changed. Only changes by other applications are reported. Example:

```c

void clip_callback(int source, void *data) {  
    if ( source == 0 ) printf("CLIP CALLBACK: selection buffer changed\n");  
    if ( source == 1 ) printf("CLIP CALLBACK: clipboard changed\n");
}  

int main() {  
    Fl::add_clipboard_notify(clip_callback);  
    \[
}
```

**Note**

Some systems require polling to monitor the clipboard and may therefore have some delay in detecting changes.

11.4.2.2 clipboard_contains()

```c

int Fl::clipboard_contains (  
    const char * type )  
```

Returns non 0 if the clipboard contains data matching `type`. The clipboard can contain both text and image data; in that situation this function returns non 0 to both requests. This function is not meant to check whether the clipboard is empty. This function does not allow to query the selection buffer because FLTK allows to copy/paste non-textual data only from/to the clipboard.
11.4.2.3 copy()

`void Fl::copy(
    const char * stuff,
    int len,
    int destination = 0,
    const char * type = Fl::clipboard_plain_text ) [static]`

Copies the data pointed to by `stuff` to the selection buffer (`destination` is 0), the clipboard (`destination` is 1), or both (`destination` is 2).

Copying to both is only relevant on X11, on other platforms it maps to the clipboard (1). `len` is the number of relevant bytes in `stuff`. `type` is always `Fl::clipboard_plain_text`. The selection buffer is used for middle-mouse pastes and for drag-and-drop selections. The clipboard is used for traditional copy/cut/paste operations.

**Note**

This function is, at present, intended only to copy UTF-8 encoded textual data. To copy graphical data, use the `Fl_Copy_Surface` class. The `type` argument may allow in the future to copy other kinds of data. Copies data to the selection buffer, the clipboard, or both.

The `destination` can be:

- 0: selection buffer (see note below)
- 1: clipboard
- 2: both

The selection buffer exists only on the X11 platform and is used for middle-mouse pastes and for drag-and-drop selections. The clipboard is used for traditional copy/cut/paste operations. On all other platforms the selection buffer (`destination = 0`) is mapped to the clipboard, i.e. on platforms other than X11 all `destination` are equivalent and the data is always copied to the clipboard.

**Note**

Please see `Fl::section_to_clipboard()` to enable duplication of the selection buffer to the clipboard on X11, i.e. if `destination = 0` (selection buffer) and `Fl::section_to_clipboard()` is enabled, then the data is copied to both the selection buffer and the clipboard. This makes the X11 behavior similar to other platforms but keeps the selection buffer for X11 specific inter process communication.

`type` should always be `Fl::clipboard_plain_text` which is the default. Other values are ignored and reserved for future extensions.

**Note**

This function is, at present, intended only to copy UTF-8 encoded textual data. To copy graphical data, use the `Fl_Copy_Surface` class. The `type` argument may allow to copy other kinds of data in the future.

**Parameters**

- **in** `stuff` text data to be copied
- **in** `len` the number of relevant bytes in `stuff`
- **in** `destination` 0 = selection, 1 = clipboard, 2 = both (see description)
- **in** `type` usually plain text (see description)
11.4.2.4  

**dnd()**

```c
int Fl::dnd ( ) [static]
```

Initiate a Drag And Drop operation.

The selection buffer should be filled with relevant data before calling this method. FLTK will then initiate the system wide drag and drop handling. Dropped data will be marked as text.

Create a selection first using: Fl::copy(const char *stuff, int len, 0)

11.4.2.5  

**paste() [1/2]**

```c
void Fl::paste ( Fl_Widget & receiver ) [static]
```

Backward compatibility only.

This calls Fl::paste(receiver, 0);

See also

Fl::paste(Fl_Widget &receiver, int clipboard, const char* type)

11.4.2.6  

**paste() [2/2]**

```c
void Fl::paste ( Fl_Widget & receiver, int source, const char *type = Fl::clipboard_plain_text ) [static]
```

Pastes the data from the selection buffer (source is 0) or the clipboard (source is 1) into receiver.

The selection buffer (source is 0) is used for middle-mouse pastes and for drag-and-drop selections. The clipboard (source is 1) is used for copy/cut/paste operations.

If source is 1, the optional type argument indicates what type of data is requested from the clipboard. At present, Fl::clipboard_plain_text (requesting text data) and Fl::clipboard_image (requesting image data) are possible. Set things up so the handle function of the receiver widget will be called with an FL_PASTE event some time in the future if the clipboard does contain data of the requested type.

The handle function of receiver can process the FL_PASTE event as follows:

- If the receiver widget is known to only receive text data, the text string from the specified source is in Fl::event_text() with UTF-8 encoding, and the number of bytes is in Fl::event_length(). If Fl::paste() gets called during the drop step of a files-drag-and-drop operation, Fl::event_text() contains a list of filenames (see Drag and Drop Events).
- If the receiver widget can potentially receive non-text data, use Fl::event_clipboard_type() to determine what sort of data is being sent. If Fl::event_clipboard_type() returns Fl::event_clipboard_plain_text, proceed as above. It it returns Fl::event_clipboard_image, the pointer returned by Fl::event_clipboard() can be safely cast to type Fl_RGB_Image * to obtain a pointer to the pasted image. If receiver accepts the clipboard image, receiver.handle() should return 1 and the application should take ownership of this image (that is, delete it after use). Conversely, if receiver.handle() returns 0, the application must not use the image.

The receiver should be prepared to be called directly by this, or for it to happen later, or possibly not at all. This allows the window system to take as long as necessary to retrieve the paste buffer (or even to screw up completely) without complex and error-prone synchronization code in FLTK.

Platform details for image data:

- Unix/Linux platform: Clipboard images in PNG or BMP formats are recognized. Requires linking with the fltk_images library.
- Windows platform: Both bitmap and vectorial (Enhanced metafile) data from clipboard can be pasted as image data.
- Mac OS X platform: Both bitmap (TIFF) and vectorial (PDF) data from clipboard can be pasted as image data.
11.4.2.7 selection()

```cpp
void Fl::selection ( 
    Fl_Widget & owner,  
    const char * text, 
    int len ) [static]
```

Changes the current selection. The block of text is copied to an internal buffer by FLTK (be careful if doing this in response to an FL_PASTE as this may be the same buffer returned by event_text()). The selection_owner() widget is set to the passed owner.

11.4.2.8 selection_owner() [1/2]

```cpp
static Fl_Widget * Fl::selection_owner ( ) [inline], [static]
```

back-compatibility only: Gets the widget owning the current selection

See also

```cpp
Fl_Widget* selection_owner(Fl_Widget*)
```

11.4.2.9 selection_owner() [2/2]

```cpp
void Fl::selection_owner ( Fl_Widget * owner ) [static]
```

Back-compatibility only: The single-argument call can be used to move the selection to another widget or to set the owner to NULL, without changing the actual text of the selection. FL_SELECTIONCLEAR is sent to the previous selection owner, if any. Copying the buffer every time the selection is changed is obviously wasteful, especially for large selections. An interface will probably be added in a future version to allow the selection to be made by a callback function. The current interface will be emulated on top of this.

11.4.2.10 selection_to_clipboard() [1/2]

```cpp
static int Fl::selection_to_clipboard ( ) [inline], [static]
```

Returns the current selection_to_clipboard mode.

See also

```cpp
void selection_to_clipboard(int)
```

11.4.2.11 selection_to_clipboard() [2/2]

```cpp
static void Fl::selection_to_clipboard ( 
    int mode ) [inline], [static]
```

Copies selections on X11 directly to the clipboard if enabled. This method can be called on all platforms. Other platforms than X11 are not affected by this feature. If this is switched on (mode = 1), Fl::copy() copies all data to the clipboard regardless of its destination argument. If the destination is 0 (selection buffer) data is copied to both the selection buffer and the clipboard. Drag and drop is also affected since drag-and-drop data is copied to the selection buffer. You can use this to make the experience of data selection and copying more like that on other platforms (Windows, macOS, and even Wayland). The default operation mode is the standard X11 behavior (disabled).

Note

This feature is experimental and enabling it may have unexpected side effects. It is your own responsibility if you enable it.

Since

1.4.0
Parameters

| in  | mode | 1 = enable selection_to_clipboard, 0 = disable selection_to_clipboard |

See also

copy(const char *, int, int, const char *)

## 11.5 Screen functions

Fl global screen functions declared in `<FL/Fl.H>`.

### Functions

- static int Fl::h ()
  - Returns the height in pixels of the main screen work area.
- static void Fl::keyboard_screen_scaling (int value)
  - Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.
- static int Fl::screen_count ()
  - Gets the total count of available screens.
- static void Fl::screen_dpi (float &h, float &v, int n=0)
  - Gets the screen resolution in dots-per-inch for the given screen.
- static int Fl::screen_num (int x, int y)
  - Gets the screen number of a screen that contains the specified screen position x, y.
- static int Fl::screen_num (int x, int y, int w, int h)
  - Gets the screen number for the screen which intersects the most with the rectangle defined by x, y, w, h.
- static float Fl::screen_scale (int n)
  - Current value of the GUI scaling factor for screen number n (n [0, Fl::screen_count()-1])
- static void Fl::screen_scale (int n, float factor)
  - Sets the value of the GUI scaling factor for screen number n (n [0, Fl::screen_count()-1]).
- static int Fl::screen_scaling_supported ()
  - See if scaling factors are supported by this platform.
- static void Fl::screen_work_area (int &X, int &Y, int &W, int &H)
  - Gets the bounding box of the work area of the screen that contains the mouse pointer.
- static void Fl::screen_work_area (int &X, int &Y, int &W, int &H, int mx, int my)
  - Gets the bounding box of the work area of a screen that contains the specified screen position mx, my.
- static void Fl::screen_work_area (int &X, int &Y, int &W, int &H, int n)
  - Gets the bounding box of the work area of the given screen.
- static void Fl::screen_xywh (int &X, int &Y, int &W, int &H)
  - Gets the bounding box of a screen that contains the mouse pointer.
- static void Fl::screen_xywh (int &X, int &Y, int &W, int &H, int mx, int my)
  - Gets the bounding box of a screen that contains the specified screen position mx, my.
- static void Fl::screen_xywh (int &X, int &Y, int &W, int &H, int mx, int my, int mw, int mh)
  - Gets the screen bounding rect for the screen which intersects the most with the rectangle defined by mx, my, mw, mh.
- static void Fl::screen_xywh (int &X, int &Y, int &W, int &H, int n)
  - Gets the screen bounding rect for the given screen.
- static int Fl::w ()
  - Returns the width in pixels of the main screen work area.
- static int Fl::x ()
  - Returns the leftmost x coordinate of the main screen work area.
- static int Fl::y ()
  - Returns the topmost y coordinate of the main screen work area.
11.5.1 Detailed Description

FL global screen functions declared in `<FL/Fl.H>`. FLTK supports high-DPI screens using a screen scaling factor. The scaling factor is initialized by the library to a value based on information obtained from the OS. If this initial value is not satisfactory, the `FLTK_SCALING_FACTOR` environment variable can be set to a value FLTK will multiply to the OS-given value. The 2 variants of functions `Fl::screen_scale()` allow to programmatically get and set scaling factor values. The scaling factor value can be further changed at runtime by typing ctrl/+/-/0/ (cmd/+/-/0/ under macOS). FLTK sends the `FL_ZOOM_EVENT` when the factor value is changed, to which a callback can be associated with `Fl::add_handler()`. By default, FLTK displays the new scaling factor value in a yellow, transient window. This can be changed with option `Fl::OPTION_SHOW_SCALING`.

11.5.2 Function Documentation

11.5.2.1 keyboard_screen_scaling()

```cpp
void Fl::keyboard_screen_scaling ( int value ) [static]
```

Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.
This function should be called before `fl_open_display()` runs. If it is not called, the default is to handle these keys for window scaling.

**Note**
This function can currently only be used to switch the internal handler off, i.e. `value` must be 0 (zero) - all other values result in undefined behavior and are reserved for future extension.

**Parameters**

| value | 0 to stop recognition of ctrl/+/-/0/ (or cmd/+/-/0/ under macOS) keys as window scaling. |

11.5.2.2 screen_count()

```cpp
int Fl::screen_count ( ) [static]
```

Gets the total count of available screens.

**Note**
Screen numbers range from 0 to `Fl::screen_count()-1` in the FLTK API.

11.5.2.3 screen_dpi()

```cpp
void Fl::screen_dpi ( float & h, float & v, int n = 0 ) [static]
```

Gets the screen resolution in dots-per-inch for the given screen.

**Parameters**

| `out` | `h,v` | horizontal and vertical resolution |
| `in`  | `n`   | the screen number (0 to `Fl::screen_count()-1`) |
11.5 Screen functions

See also

```c
void screen_xywh(int &x, int &y, int &w, int &h, int mx, int my)
```

11.5.2.4 screen_num() [1/2]

```c
int Fl::screen_num ( int x, int y ) [static]
```

Gets the screen number of a screen that contains the specified screen position \( x, y \).

**Parameters**

- **\( x, y \)**: the absolute screen position

**Returns**

a screen number \([0, Fl::screen_count()-1]\)

**Attention**

When the running system contains screens with different scaling factor values, this API may become ambiguous because a given value pair \((x, y)\) may belong to distinct screens. In that situation, other APIs should be preferred, e.g., \( Fl\_Window::screen_num() \) and \( Fl::screen_scale(int) \).

11.5.2.5 screen_num() [2/2]

```c
int Fl::screen_num ( int x, int y, int w, int h ) [static]
```

Gets the screen number for the screen which intersects the most with the rectangle defined by \( x, y, w, h \).

**Parameters**

- **\( x, y, w, h \)**: the rectangle to search for intersection with

**Returns**

a screen number \([0, Fl::screen_count()-1]\)

11.5.2.6 screen_scale()

```c
void Fl::screen_scale ( int n, float factor ) [static]
```

Sets the value of the GUI scaling factor for screen number \( n \) \((n \in [0, Fl::screen_count()-1])\).

Also sets the scale factor value of all windows mapped to screen number \( n \), if any.

11.5.2.7 screen_scaling_supported()

```c
int Fl::screen_scaling_supported ( ) [static]
```

See if scaling factors are supported by this platform.
Returns
0 if scaling factors are not supported by this platform, 1 if a single scaling factor value is shared by all screens,
2 if each screen can have its own scaling factor value.

See also
Fl::screen_scale(int)

11.5.2.8 screen_work_area() [1/3]

```c
void Fl::screen_work_area (  
    int & X,  
    int & Y,  
    int & W,  
    int & H ) [static]
```

Gets the bounding box of the work area of the screen that contains the mouse pointer.

Parameters
- **out X,Y,W,H** the work area bounding box

See also
```
void screen_work_area(int &x, int &y, int &w, int &h, int mx, int my)
```

11.5.2.9 screen_work_area() [2/3]

```c
void Fl::screen_work_area (  
    int & X,  
    int & Y,  
    int & W,  
    int & H,  
    int mx,  
    int my ) [static]
```

Gets the bounding box of the work area of a screen that contains the specified screen position mx, my.

Parameters
- **out X,Y,W,H** the work area bounding box
- **in mx,my** the absolute screen position

11.5.2.10 screen_work_area() [3/3]

```c
void Fl::screen_work_area (  
    int & X,  
    int & Y,  
    int & W,  
    int & H,  
    int n ) [static]
```

Gets the bounding box of the work area of the given screen.

Parameters
- **out X,Y,W,H** the work area bounding box
- **in n** the screen number (0 to Fl::screen_count() - 1)
11.5 Screen functions

See also

```c
void screen_xywh(int &x, int &y, int &w, int &h, int mx, int my)
```

Note

Like all quantities accessible via public APIs of FLTK, values of $X, Y, W, H$ are given in FLTK units, that is, in drawing units divided by the scaling factor of screen $n$.

11.5.2.11 screen_xywh() [1/4]

```c
void Fl::screen_xywh(
    int & X, 
    int & Y, 
    int & W, 
    int & H ); [static]
```

Gets the bounding box of a screen that contains the mouse pointer.

Parameters

| out  | X, Y, W, H | the corresponding screen bounding box |

See also

```c
void screen_xywh(int &x, int &y, int &w, int &h, int mx, int my)
```

11.5.2.12 screen_xywh() [2/4]

```c
void Fl::screen_xywh(
    int & X, 
    int & Y, 
    int & W, 
    int & H, 
    int mx, 
    int my ); [static]
```

Gets the bounding box of a screen that contains the specified screen position $mx, my$.

Parameters

| out  | X, Y, W, H | the corresponding screen bounding box |
| in   | mx, my     | the absolute screen position          |

11.5.2.13 screen_xywh() [3/4]

```c
void Fl::screen_xywh(
    int & X, 
    int & Y, 
    int & W, 
    int & H, 
    int mx, 
    int my, 
    int mw, 
    int mh ); [static]
```

Gets the screen bounding rect for the screen which intersects the most with the rectangle defined by $mx, my, mw, mh$. 

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>X, Y, W, H</th>
<th>the corresponding screen bounding box</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mx, my, mw, mh</td>
<td>the rectangle to search for intersection with</td>
</tr>
</tbody>
</table>

See also

void screen_xywh(int &X, int &Y, int &W, int &H, int n)

11.5.2.14 screen_xywh() [4/4]

void Fl::screen_xywh (  
    int & X,  
    int & Y,  
    int & W,  
    int & H,  
    int n ) [static]

Gets the screen bounding rect for the given screen.
Under Windows, Mac OS X, and X11 + the Gnome desktop, screen #0 contains the menubar/taskbar

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>X, Y, W, H</th>
<th>the corresponding screen bounding box</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>the screen number (0 to Fl::screen_count() - 1)</td>
</tr>
</tbody>
</table>

Note

Like all quantities accessible via public APIs of FLTK, values of X,Y,W,H are given in FLTK units, that is, in
drawing units divided by the scaling factor of screen n.

See also

void screen_xywh(int &x, int &y, int &w, int &h, int mx, int my)

11.6 Color & Font functions

fl global color, font functions.

Functions

• Fl_Color fl_color ()  
  Return the last fl_color() that was set.
• void fl_color (Fl_Color c)  
  Set the color for all subsequent drawing operations.
• void fl_color (int c)  
  for back compatibility - use fl_color(Fl_Color c) instead
• void fl_color (uchar r, uchar g, uchar b)  
  Set the color for all subsequent drawing operations.
• Fl_Color fl_color_average (Fl_Color color1, Fl_Color color2, float weight)  
  Returns the weighted average color between the two given colors.
• Fl_Color fl_contrast (Fl_Color fg, Fl_Color bg, Fl_Fontsize fs, int context)  
  Returns a color that contrasts with the background color.
• void fl_contrast_function (Fl_Contrast_Function *f)  
  Register a custom contrast function.
11.6 Color & Font functions

- **int fl_contrast_level ()**
  
  Get the contrast level (sensitivity) of the \texttt{fl_contrast()} method.

- **void \texttt{fl_contrast_level} (int level)**
  
  Set the contrast level (sensitivity) of the \texttt{fl_contrast()} method.

- **int fl_contrast_mode ()**
  
  Return the current contrast algorithm (mode).

- **void \texttt{fl_contrast_mode} (int mode)**
  
  Set the contrast algorithm (mode).

- **int fl_descent ()**
  
  Return the recommended distance above the bottom of a \texttt{fl_height()} tall box to draw the text at so it looks centered vertically in that box.

- **\texttt{Fl_Font fl_font ()}**
  
  Return the face set by the most recent call to \texttt{fl_font()}.  

- **void \texttt{fl_font} (\texttt{Fl_Font face}, \texttt{Fl_Fontsize fsize})**
  
  Sets the current font, which is then used in various drawing routines.

- **int \texttt{fl_height ()}**
  
  Return the recommended minimum line spacing for the current font.

- **int \texttt{fl_height} (int font, int size)**
  
  This function returns the actual height of the specified font and size.

- **\texttt{Fl_Color fl_inactive} (\texttt{Fl_Color c})**
  
  Returns the inactive, dimmed version of the given color.

- **const char ∗ \texttt{fl_latin1_to_local} (const char ∗t, int n=-1)**
  
  Convert text from Windows/X11 latin1 character set to local encoding.

- **double \texttt{fl_lightness} (\texttt{Fl_Color color})**
  
  Return the perceived lightness of a color.

- **const char ∗ \texttt{fl_local_to_latin1} (const char ∗t, int n=-1)**
  
  Convert text from local encoding to Windows/X11 latin1 character set.

- **const char ∗ \texttt{fl_local_to_mac_roman} (const char ∗t, int n=-1)**
  
  Convert text from local encoding to Mac Roman character set.

- **double \texttt{fl_luminance} (\texttt{Fl_Color color})**
  
  Return the raw / physical luminance of a color.

- **const char ∗ \texttt{fl_mac_roman_to_local} (const char ∗t, int n=-1)**
  
  Convert text from Mac Roman character set to local encoding.

- **\texttt{Fl_Color fl_show_colormap} (\texttt{Fl_Color oldcol})**
  
  Pops up a window to let the user pick a colormap entry.

- **\texttt{Fl_Fontsize fl_size ()}**
  
  Return the size set by the most recent call to \texttt{fl_font()}.  

- **void \texttt{fl_text_extents} (const char ∗, int &dx, int &dy, int &w, int &h)**
  
  Determine the minimum pixel dimensions of a nul-terminated string using the current \texttt{fl_font()}.  

- **void \texttt{fl_text_extents} (const char ∗t, int n, int &dx, int &dy, int &w, int &h)**
  
  Determine the minimum pixel dimensions of a sequence of \texttt{n} characters (bytes) using the current \texttt{fl_font()}.  

- **double \texttt{fl_width} (const char ∗t)**
  
  Return the typographical width of a nul-terminated string using the current font face and size.

- **double \texttt{fl_width} (const char ∗t, int n)**
  
  Return the typographical width of a sequence of \texttt{n} characters using the current font face and size.

- **double \texttt{fl_width} (unsigned int c)**
  
  Return the typographical width of a single character using the current font face and size.

- **static void \texttt{Fl::free_color} (\texttt{Fl_Color i}, int overlay=0)**
  
  Frees the specified color from the colormap, if applicable.

- **static unsigned \texttt{Fl::get_color} (\texttt{Fl_Color i})**
  
  Returns the RGB value(s) for the given FLTK color index.
• static void Fl::get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue)
   
   Returns the RGB value(s) for the given FLTK color index.

• static void Fl::get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue, uchar &alpha)
   
   Returns the RGBA value(s) for the given FLTK color index.

• static const char * Fl::get_font (Fl_Font)
   
   Gets the string for this face.

• static const char * Fl::get_font_name (Fl_Font, int *attributes=0)
   
   Get a human-readable string describing the family of this face.

• static int Fl::get_font_sizes (Fl_Font, int *sizep)
   
   Return an array of sizes in sizep.

• static void Fl::set_color (Fl_Color i, uchar, uchar, uchar)
   
   Sets an entry in the fl_color index table.

• static void Fl::set_color (Fl_Color, uchar, uchar, uchar, uchar)
   
   Sets an entry in the fl_color index table.

• static void Fl::set_font (Fl_Font, const char *)
   
   Changes a face.

• static void Fl::set_font (Fl_Font, Fl_Font)
   
   Copies one face to another.

• static Fl_Font Fl::set_fonts (const char * =0)
   
   FLTK will open the display, and add every fonts on the server to the face table.

11.6.1 Detailed Description

Fl global color, font functions.
These functions are declared in `<FL/Fl.H>` or `<FL/fl_draw.H>`.

11.6.2 Function Documentation

11.6.2.1 fl_color() [1/3]

Fl_Color fl_color () [inline]

Return the last fl_color() that was set.
This can be used for state save/restore.

11.6.2.2 fl_color() [2/3]

void fl_color ( Fl_Color c ) [inline]

Set the color for all subsequent drawing operations.
For color-mapped displays, a color cell will be allocated out of fl_colormap the first time you use a color. If the colormap fills up then a least-squares algorithm is used to find the closest color. If no valid graphical context (fl_gc) is available, the foreground is not set for the current window.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>c</td>
<td>color</td>
</tr>
</tbody>
</table>

11.6.2.3 fl_color() [3/3]

void fl_color ( uchar r,
    uchar g,
uchar b ) [inline]

Set the color for all subsequent drawing operations.
The closest possible match to the RGB color is used. The RGB color is used directly on TrueColor displays. For
colormap visuals the nearest index in the gray ramp or color cube is used. If no valid graphical context (fl_gc) is
available, the foreground is not set for the current window.

Parameters

| in | r,g,b | color components |

11.6.2.4 fl_color_average()

Fl_Color fl_color_average ( Fl_Color color1,
                        Fl_Color color2,
                        float weight )

Returns the weighted average color between the two given colors.
The red, green and blue values are averages using the following formula:

\[
\text{color} = \text{color1} \times \text{weight} + \text{color2} \times (1 - \text{weight})
\]

Thus, a weight value of 1.0 will return the first color, while a value of 0.0 will return the second color.

Parameters

| in | color1,color2 | boundary colors |
|    | weight         | weighting factor |

11.6.2.5 fl_contrast()

Fl_Color fl_contrast ( Fl_Color fg,
                     Fl_Color bg,
                     Fl_Fontsize fs,
                     int context )

Returns a color that contrasts with the background color.
This will be the foreground color if it contrasts sufficiently with the background color. Otherwise, returns FL_WHITE
or FL_BLACK depending on which color provides the best contrast.

FLTK 1.4.0 uses a different default contrast function than earlier releases (1.3.x) but you can use the old "legacy"
contrast function by calling

fl_contrast_mode(FL_CONTRAST_LEGACY);
early in your main program.

Note

It is a known issue that static initialization using fl_contrast() may already have been executed before you
call this function in main(). You should be aware of this and, if necessary, write your own (static) contrast
initialization function. This should rarely be necessary.

You can change the behavior of fl_contrast() in several ways:

- Change the "level" (sensitivity) for contrast calculation, see fl_contrast_level(). Valid levels are 0 - 100, the
default "medium" value is 50. If you raise the level above 50 the overall contrast will generally be higher, i.e.
the required contrast to return the foreground color is raised and therefore the calculated color will switch
"earlier" to either black or white. In other words, using the following values:

  - 0 will always use the foreground color
  - 50 will use the default, unmodified algorithm
  - 100 will always use black or white
values larger than 50 may yield slightly better results. Changing the level is particularly useful and intended for the "legacy mode" to improve the results partially. Values slightly above 50 (50 - 70) will likely return the best results (50 is the default, as used in FLTK 1.3.x).

Note

Different contrast modes (algorithms) can use their own values and defaults of fl_contrast_level().

• Change the used contrast calculation function. You can either use the old (FLTK 1.3.x) function or use the better but slower function based on the CIELAB (L*a*b*) color model, or you can define your own custom contrast function if you need even better contrast results.

The following contrast functions are available:

• FL_CONTRAST_LEGACY, the old FLTK 1.3.x compatible function. This is the fastest function (using integer arithmetic) but it provides worse results in border cases. You may want to use this on embedded or otherwise CPU constrained systems or if you need strict backwards compatibility. For slightly better results you may utilize the new fl_contrast_level(int) function (since 1.4.0) to increase the contrast sensitivity. This will provide slightly better results than FLTK 1.3.x and earlier but we recommend to use the new default function:

• FL_CONTRAST_CIELAB, based on the CIELAB (L*a*b*) color model. This function is superior regarding the visual contrast perception but may be slower. This is the default since FLTK 1.4.0.

• FL_CONTRAST_CUSTOM, your own contrast calculation function.

In the future we may provide even more (and superior) contrast algorithms.

The new parameters fs and context (since 1.4.0) are defined for future extensions and are currently not used. Default values are 0.

Note

These new optional parameters must be provided in the custom contrast function which is the reason why they are added now. In the future we may use the fontsize to adjust the calculated contrast, and users defining their own contrast functions may use them in their functions.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fg</th>
<th>foreground (text) color</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bg</td>
<td>background color</td>
</tr>
<tr>
<td>in</td>
<td>fs</td>
<td>font size (optional, default = 0 == undefined)</td>
</tr>
<tr>
<td>in</td>
<td>context</td>
<td>graphical context (optional, default = 0 == text)</td>
</tr>
</tbody>
</table>

Returns

contrasting color: fg, FL_BLACK, or FL_WHITE

See also

fl_contrast_level(int)
fl_contrast_mode(int)
fl_contrast_function()
Your custom contrast function must provide the signature

```cpp
FL_Color my_contrast_function(FL_Color fg, FL_Color bg, Fl_Fontsize fs, int context)
```

The arguments are the same as for the full `fl_contrast()` function since FLTK 1.4. You can use the supplied fontsize `fs` to modify the result. Depending on the caller the `fs` parameter can be 0 (default) or a valid fontsize. The `context` parameter is not yet used and will always be 0 unless included in a call to `fl_contrast()`. The value 0 should be interpreted as text. In the future the `context` argument will be used to supply a different context than text (small icons, large icons, etc.). The exact usage is not yet specified.

Your function may also use `fl_contrast_level()` to modify the result accordingly.

Since

1.4.0

See also

- `fl_contrast_mode(int)`
- `fl_contrast_level(int)`
- `fl_contrast()`

### 11.6.2.7 fl_contrast_level() [1/2]

```cpp
int fl_contrast_level ( )
```

Get the contrast level (sensitivity) of the `fl_contrast()` method.

This returns the level of the currently selected contrast mode.

Returns

The current contrast level.

See also

- `fl_contrast_level(int level)`
- `fl_contrast_mode(int mode)`

Since

1.4.0

### 11.6.2.8 fl_contrast_level() [2/2]

```cpp
void fl_contrast_level ( int level )
```

Set the contrast level (sensitivity) of the `fl_contrast()` method.

This can be used to tune the legacy `fl_contrast()` function to achieve slightly better results. The default value is defined per contrast mode (see below). Values between 50 and 70 are recommended but you can raise it up to 100. Lower values than 50 are probably not useful.

The contrast level affects not only the legacy (1.3.x) `fl_contrast()` function but also the new CIELAB contrast mode which is the default since FLTK 1.4.0.

Other contrast modes are currently not affected by the contrast level.

You may use the contrast level if you define your own custom contrast function in mode `FL_CONTRAST_CUSTOM`.

Note

All contrast modes store their own contrast level because the behavior is slightly different. You must change the contrast mode `fl_contrast_mode()` **before** you set or get the contrast level.

The default contrast level is

- 50 in mode `FL_CONTRAST_LEGACY` (compatible with FLTK 1.3.x)
- 55 in mode `FL_CONTRAST_CIELAB`
• 0 (undefined) for all other modes

See the description of `fl_contrast_mode(int mode)` for more information about the contrast level per mode.

Example:
```
fl_contrast_mode(FL_CONTRAST_LEGACY);
fl_contrast_level(65);
```

A level greater than 50 (probably best in the range 50 to 70) may achieve better results of the legacy `fl_contrast()` function in some border cases of low contrast between foreground and background colors but we recommend to use the new default algorithm `FL_CONTRAST_CIELAB` unless you need strict backwards compatibility or use a CPU constrained embedded system.

Parameters

| in | level | valid range is 0 to 100 |

Since

1.4.0

11.6.2.9  `fl_contrast_mode()` [1/2]

```cpp
int fl_contrast_mode()
```

Return the current contrast algorithm (mode).

Returns

Contrast algorithm (mode).

Since

1.4.0

See also

`fl_contrast_mode(int)`

11.6.2.10  `fl_contrast_mode()` [2/2]

```cpp
void fl_contrast_mode(int mode)
```

Set the contrast algorithm (mode).

You can use one of

• FL_CONTRAST_NONE (not recommended: returns the foreground color)
• FL_CONTRAST_LEGACY (same as in FLTK 1.3.x)
• FL_CONTRAST_CIELAB (default since FLTK 1.4.0)
• FL_CONTRAST_CUSTOM (you must define your own contrast algorithm)

If you set FL_CONTRAST_CUSTOM you must also register your custom contrast function by calling `fl_contrast_function()`.

You may set the contrast level `fl_contrast_level(int)` after setting the contrast mode. This affects the contrast algorithm as described below:

• FL_CONTRAST_LEGACY: default level is 50 which is compatible with FLTK 1.3.x and older. This mode is no longer the default and is not recommended because it doesn't take human contrast perception into account and doesn't properly handle sRGB color values. You may get better contrasts if you set the level higher than 50. Values in the range 50 to 70 may be useful. Higher values result in higher contrast, i.e. the algorithm switches “earlier” to black or white mode.
• FL_CONTRAST_CIELAB: default level is 55 which appears to be a good value. The higher the level is, the more contrast is to be expected. Values in the range below 55 accept lower contrast and values above 55 switch “earlier” to black or white. Values between 45 and 65 may yield usable contrast experience.
11.6 Color & Font functions

Parameters

\[ \text{in } \text{mode} \quad \text{if invalid, FL\_CONTRAST\_CIELAB will be selected} \]

Since

1.4.0

See also

\begin{verbatim}
fl_contrast_function(Fl_ContrastFunction *)
fl_contrast_level(int)
\end{verbatim}

11.6.2.11 fl_font() [1/2]

\begin{verbatim}
Fl_Font fl_font () \{inline\}
\end{verbatim}

Return the face set by the most recent call to fl_font(). This can be used to save/restore the font.

11.6.2.12 fl_font() [2/2]

\begin{verbatim}
void fl_font ( Fl_Font face, Fl_Fontsize fsize )
\end{verbatim}

Sets the current font, which is then used in various drawing routines. You may call this outside a draw context if necessary to measure text, for instance by calling fl_width(), fl_measure(), or fl_text_extents(), but on X this will open the display.

The font is identified by a face and a size. The size of the font is measured in pixels and not "points". Lines should be spaced \texttt{size} pixels apart or more.

11.6.2.13 fl_height() [1/2]

\begin{verbatim}
int fl_height () \{inline\}
\end{verbatim}

Return the recommended minimum line spacing for the current font. You can also use the value of \texttt{size} passed to fl_font().

11.6.2.14 fl_height() [2/2]

\begin{verbatim}
int fl_height ( int font, int size )
\end{verbatim}

This function returns the actual height of the specified font and size. Normally the font height should always be 'size', but with the advent of XFT, there are (currently) complexities that seem to only be solved by asking the font what its actual font height is. (See STR#2115)

This function was originally undocumented in 1.1.x, and was used only by Fl_Text_Display. We're now documenting it in 1.3.x so that apps that need precise height info can get it with this function.

Returns

the height of the font in pixels.

\textbf{Todo} In the future, when the XFT issues are resolved, this function should simply return the 'size' value.

11.6.2.15 fl_latin1_to_local()

\begin{verbatim}
const char * fl_latin1_to_local ( const char * t, int n = -1 )
\end{verbatim}

Convert text from Windows/X11 latin1 character set to local encoding.
11.6.2.16  fl_lightness()

double fl_lightness ( 
    Fl_Color color )

Return the perceived lightness of a color.
This function calculates the perceived lightness of Fl_Color color.
The returned lightness value $L*$ according to the CIELAB ($L^*a^*b^*$) color model is almost linear with respect to human perception. It is in the range 0 (black) to 100 (white).
The result values of two colors can be compared directly and the difference is their perceived contrast.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>color</th>
<th>Fl_Color value</th>
</tr>
</thead>
</table>

Returns

perceived lightness (0 .. 100)

Since

1.4.0

11.6.2.17  fl_local_to_latin1()

const char* fl_local_to_latin1 ( 
    const char* t, 
    int n = -1 )

Convert text from local encoding to Windows/X11 latin1 character set.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>character string (local encoding)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>optional number of characters (bytes) to convert (default is all)</td>
</tr>
</tbody>
</table>

Returns

pointer to internal buffer containing converted characters

11.6.2.18  fl_local_to_mac_roman()

const char* fl_local_to_mac_roman ( 
    const char* t, 
    int n = -1 )

Convert text from local encoding to Mac Roman character set.
11.6 Color & Font functions

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>character string (local encoding)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>optional number of characters to convert (default is all)</td>
</tr>
</tbody>
</table>

Returns

pointer to internal buffer containing converted characters

11.6.2.19  fl_luminance()

double fl_luminance ( 
    Fl_Color color )

Return the raw / physical luminance of a color.
This function calculates the physical luminance of Fl_Color color.
The returned luminance value (aka \(Y\)) is the physical luminance of the Fl_Color color.
The result is in the range 0.0 (black) to 1.0 (white).

Note

This is probably not what you want if you are interested in perceived contrast or lightness calculation because
the luminance \(Y\) is not linear with respect to human perception.

See fl_lightness(Fl_Color) for a function that returns the perceived lightness of a color which can be used directly
for contrast calculation.

Parameters

| in | color | Fl_Color value |

Returns

Raw (physical) luminance (0.0 .. 1.0)

Since

1.4.0

See also

fl_lightness(Fl_Color)

11.6.2.20  fl_mac_roman_to_local()

const char * fl_mac_roman_to_local ( 
    const char * t, 
    int n = -1 )

Convert text from Mac Roman character set to local encoding.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>character string (Mac Roman encoding)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>optional number of characters to convert (default is all)</td>
</tr>
</tbody>
</table>
Returns

pointer to internal buffer containing converted characters

11.6.21  fl_show_colormap()

\texttt{Fl\_Color fl\_show\_colormap (}
\texttt{    Fl\_Color \textit{oldcol} )}

Pops up a window to let the user pick a colormap entry.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fl_show_colormap.png}
\caption{fl\_show\_colormap}
\end{figure}

Parameters

\begin{tabular}{|c|c|}
\hline
\textbf{in} & \textit{oldcol} & color to be highlighted when grid is shown. \\
\hline
\end{tabular}

Return values

\begin{tabular}{|c|c|}
\hline
\textit{Fl\_Color} & value of the chosen colormap entry. \\
\hline
\end{tabular}

See also

\texttt{Fl\_Color\_Chooser}

11.6.22  fl\_size()

\texttt{Fl\_Fontsize fl\_size ( ) [inline]}

Return the size set by the most recent call to \texttt{fl\_font()}. This can be used to save/restore the font.
11.6.2.23  fl_text_extents()  [1/2]

void fl_text_extents (  
    const char * c,  
    int & dx,  
    int & dy,  
    int & w,  
    int & h  
)  

Determine the minimum pixel dimensions of a nul-terminated string using the current fl_font().
Usage: given a string "txt" drawn using fl_draw(txt, x, y) you would determine its pixel extents on the display using
fl_text_extents(txt, dx, dy, wo, ho) such that a bounding box that exactly fits around the text could be drawn with
fl_rect(x+dx, y+dy, wo, ho). Note the dx, dy values hold the offset of the first "colored in" pixel of the string, from the
draw origin.
Note the desired font and font size must be set with fl_font() before calling this function.
This differs slightly from fl_measure() in that the dx/dy values are also returned.
No FLTK symbol expansion will be performed.
Example use:
int dx,dy,W,H;
fl_font(FL_HELVETICA, 12); // set font face+size first
fl_text_extents("Some text", dx, dy, W, H); // get width and height of string
printf("text's width=%d, height=%d
", W, H);

11.6.2.24  fl_text_extents()  [2/2]

void fl_text_extents (  
    const char * t,  
    int n,  
    int & dx,  
    int & dy,  
    int & w,  
    int & h   
)  [inline]  

Determine the minimum pixel dimensions of a sequence of n characters (bytes) using the current fl_font().
Note
The string length is measured in bytes, not (UTF-8) characters.
See also
    fl_text_extents(const char*, int& dx, int& dy, int& w, int& h)

11.6.2.25  fl_width()  

double fl_width (  
    unsigned int c  
)  [inline]  

Return the typographical width of a single character using the current font face and size.
Note
If a valid fl_gc is NOT found then it uses the first window gc, or the screen gc if no fltk window is available
when called.

11.6.2.26  free_color()  

void Fl::free_color (  
    Fl_Color i,  
    int overlay = 0    
)  [static]  

Frees the specified color from the colormap, if applicable.
If overlay is non-zero then the color is freed from the overlay colormap.
### 11.6.2.27 \texttt{get\_color()} [1/3]

```c
unsigned Fl::get\_color ( Fl\_Color i ) [static]
```

Returns the RGB value(s) for the given FLTK color index. This form returns the RGB values packed in a 32-bit unsigned integer with the red value in the upper 8 bits, the green value in the next 8 bits, and the blue value in bits 8-15. The lower 8 bits will always be 0.

### 11.6.2.28 \texttt{get\_color()} [2/3]

```c
void Fl::get\_color ( Fl\_Color i,
                     uchar \& red,
                     uchar \& green,
                     uchar \& blue ) [static]
```

Returns the RGB value(s) for the given FLTK color index. This form returns the red, green, and blue values separately in referenced variables.

See also

```c
unsigned get\_color(Fl\_Color c)
```

### 11.6.2.29 \texttt{get\_color()} [3/3]

```c
void Fl::get\_color ( Fl\_Color i,
                     uchar \& red,
                     uchar \& green,
                     uchar \& blue,
                     uchar \& alpha ) [static]
```

Returns the RGBA value(s) for the given FLTK color index. This form returns the red, green, blue, and alpha values separately in referenced variables.

See also

```c
unsigned get\_color(Fl\_Color c)
```

### 11.6.30 \texttt{get\_font()} 

```c
const char \* Fl::get\_font ( Fl\_Font fnum ) [static]
```

Gets the string for this face. This string is different for each face. Under X this value is passed to XListFonts to get all the sizes of this face.

### 11.6.31 \texttt{get\_font\_name()}

```c
const char \* Fl::get\_font\_name ( Fl\_Font fnum,
                                   int \* attributes = 0 ) [static]
```

Get a human-readable string describing the family of this face. This is useful if you are presenting a choice to the user. There is no guarantee that each face has a different name. The return value points to a static buffer that is overwritten each call. The integer pointed to by \texttt{attributes} (if the pointer is not zero) is set to zero, FL\_BOLD or FL\_ITALIC or FL\_BOLD | FL\_ITALIC. To locate a “family” of fonts, search forward and back for a set with non-zero attributes, these faces along with the face with a zero attribute before them constitute a family.

### 11.6.32 \texttt{get\_font\_sizes()}

```c
int Fl::get\_font\_sizes ( Fl\_Font fnum,
                          int \& sizep ) [static]
```

Generated by Doxygen
Return an array of sizes in `sizep`. The return value is the length of this array. The sizes are sorted from smallest to largest and indicate what sizes can be given to `fl_font()` that will be matched exactly (`fl_font()` will pick the closest size for other sizes). A zero in the first location of the array indicates a scalable font, where any size works, although the array may list sizes that work “better” than others. Warning: the returned array points at a static buffer that is overwritten each call. Under X this will open the display.

### 11.6.2.33 set_color() [1/3]

```c
void Fl::set_color ( 
  Fl_Color i, 
  unsigned c ) [static]
```

Sets an entry in the `fl_color` index table.
You can set it to any 8-bit RGB color. The color is not allocated until `fl_color(i)` is used.

### 11.6.2.34 set_color() [2/3]

```c
void Fl::set_color ( 
  Fl_Color i, 
  uchar red, 
  uchar green, 
  uchar blue ) [static]
```

Sets an entry in the `fl_color` index table.
You can set it to any 8-bit RGB color. The color is not allocated until `fl_color(i)` is used.

### 11.6.2.35 set_color() [3/3]

```c
void Fl::set_color ( 
  Fl_Color i, 
  uchar red, 
  uchar green, 
  uchar blue, 
  uchar alpha ) [static]
```

Sets an entry in the `fl_color` index table.
You can set it to any 8-bit RGBA color.

**Note**

The color transparency is effective under the Wayland, hybrid Wayland/X11 and macOS platforms, whereas it has no effect under the X11 and Windows platforms. It's also effective for widgets added to an `Fl_Gl_Window`.

**Version**

1.4

### 11.6.2.36 set_font()

```c
void Fl::set_font ( 
  Fl_Font fnum, 
  const char * name ) [static]
```

Changes a face.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fnum</code></td>
<td>The font number to be assigned a new face</td>
</tr>
<tr>
<td><code>name</code></td>
<td>Name of the font to assign. The string pointer is simply stored, the string is not copied, so the string must be in static memory. The exact name to be used depends on the platform :</td>
</tr>
</tbody>
</table>
• Windows, X11, Xft: use the family name prefixed by one character to indicate the desired font variant. Characters ' ', 'I', 'B', 'P' denote plain, italic, bold, and bold-italic variants, respectively. For example, string "IGabriola" is to be used to denote the "Gabriola italic" font. The "Oblique" suffix, in whatever case, is to be treated as "italic", that is, prefix the family name with 'I'.

• Other platforms, i.e., X11 + Pango, Wayland, macOS: use the full font name as returned by function Fl::get_font_name() or as listed by applications test/fonts or test/utf8. No prefix is to be added.

11.6.2.37 set_fonts()

Fl_Font Fl::set_fonts (const char *xstarname = 0) [static]

FLTK will open the display, and add every fonts on the server to the face table. It will attempt to put "families" of faces together, so that the normal one is first, followed by bold, italic, and bold italic. The only argument to this function is somewhat obsolete since FLTK and most underlying operating systems move to support Unicode. For completeness, following is the original documentation and a few updates:

On X11, the optional argument is a string to describe the set of fonts to add. Passing NULL will select only fonts that have the ISO8859-1 character set (and are thus usable by normal text). Passing "-*" will select all fonts with any encoding as long as they have normal X font names with dashes in them. Passing "*" will list every font that exists (on X this may produce some strange output). Other values may be useful but are system dependent.

With the Xft option on Linux, this parameter is ignored.

With Windows, NULL selects fonts with ANSI_CHARSET encoding and non-NULL selects all fonts. On macOS, this parameter is ignored.

The return value is how many faces are in the table after this is done.

11.7 Drawing functions

FLTK global graphics and GUI drawing functions.

Enumerations

• enum {
  FL_SOLID = 0, FL_DASH = 1, FL_DOT = 2, FL_DASHDOT = 3,
  FL_DASHDOTDOT = 4, FL_CAP_FLAT = 0x100, FL_CAP_ROUND = 0x200, FL_CAP_SQUARE = 0x300,
  FL_JOIN_MITER = 0x1000, FL_JOIN_ROUND = 0x2000, FL_JOIN_BEVEL = 0x3000
}

Functions

• int fl_add_symbol (const char *name, void(*drawit)(Fl_Color), int scalable)
  Adds a symbol to the system.

• int fl_antialias ()
  Return whether line drawings are currently antialiased.

• void fl_antialias (int state)
  Turn antialiased line drawings ON or OFF, if supported by platform.

• void fl_arc (double x, double y, double r, double start, double end)
  Add a series of points to the current path on the arc of a circle.

• void fl_arc (int x, int y, int w, int h, double a1, double a2)
  Draw ellipse sections using integer coordinates.

• void fl_begin_complex_polygon ()
  Start drawing a complex filled polygon.

• void fl_begin_line ()
  Start drawing a list of lines.

• void fl_begin_loop ()
  Start drawing a closed sequence of lines.

• void fl_begin_offscreen (Fl_Offscreen ctx)
Send all subsequent drawing commands to this offscreen buffer.

- **void fl_begin_points ()**
  Start drawing a list of points.

- **void fl_begin_polygon ()**
  Start drawing a convex filled polygon.

- **char fl_can_do_alpha_blending ()**
  Check whether platform supports true alpha blending for RGBA images.

- **Fl_RGB_Image * fl_capture_window (Fl_Window *win, int x, int y, int w, int h)**
  Captures the content of a rectangular zone of a mapped window.

- **void fl_chord (int x, int y, int w, int h, double a1, double a2)**
  *fl_chord declaration is a place holder - the function does not yet exist*

- **void fl_circle (double x, double y, double r)**
  *fl_circle(x,y,r) is equivalent to fl_arc(x,y,r,0,360), but may be faster.*

- **void fl_clip (int x, int y, int w, int h)**
  Intersect the current clip region with a rectangle and push this new region onto the stack (deprecated).

- **int fl_clip_box (int x, int y, int w, int h, int &X, int &Y , int &W, int &H)**
  Intersect a rectangle with the current clip region and return the bounding box of the result.

- **Fl_Region fl_clip_region ()**
  Return the current clipping region.

- **void fl_clip_region (Fl_Region r)**
  Replace the top of the clipping stack with a clipping region of any shape.

- **void fl_copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)**
  Copy a rectangular area of the given offscreen buffer into the current drawing destination.

- **Fl_Offscreen fl_create_offscreen (int w, int h)**
  Creation of an offscreen graphics buffer.

- **void fl_cursor (Fl_Cursor)**
  Sets the cursor for the current window to the specified shape and colors.

- **void fl_cursor (Fl_Cursor, Fl_Color fg, Fl_Color bg=FL_WHITE)**
  Add a series of points on a Bézier curve to the path.

- **void fl_curve (double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)**
  Add a series of points on a Bézier curve to the path.

- **void fl_delete_offscreen (Fl_Offscreen ctx)**
  Deletion of an offscreen graphics buffer.

- **void fl_draw (const char *str, int n, int x, int y)**
  Draws starting at the given x, y location a UTF-8 string of length n bytes.

- **void fl_draw (const char *str, int x, int y)**
  Draws a nul-terminated UTF-8 string starting at the given x, y location.

- **void fl_draw (const char *str, int x, int y, int w, int h, Fl_Align align, Fl_Image *img=0, int draw_symbols=1)**
  Fancy string drawing function which is used to draw all the labels.

- **void fl_draw (const char *str, int x, int y, int w, int h, Fl_Align align, Fl_Image *img=0, int draw_symbols=1)**
  The same as fl_draw(const char *,int,int,int,Fl_Align,Fl_Image *,int) with the addition of the *callthis* parameter, which is a pointer to a text drawing function such as fl_draw(const char *, int, int, int) to do the real work.

- **void fl_draw (int angle, const char *str, int n, int x, int y)**
  Draws at the given x, y location a UTF-8 string of length n bytes rotating angle degrees counter-clockwise.

- **void fl_draw (int angle, const char *str, int x, int y)**
  Draws a nul-terminated UTF-8 string starting at the given x, y location and rotating angle degrees counter-clockwise.

- **void fl_draw_arrow (Fl_Rect bb, Fl_Arrow_Type t, Fl_Orientation o, Fl_Color color)**
  Draws an "arrow like" GUI element for the selected scheme.

- **void fl_draw_box (Fl_Boxtype, int x, int y, int w, int h, Fl_Color)**
  Draws a box using given type, position, size and color.
Draw a check mark inside the given bounding box.

- **void fl_draw_circle (int x, int y, int d, Fl_Color color)**
  Draw a potentially small, filled circle using a given color.

- **void fl_draw_image (const uchar *buf, int X, int Y, int W, int H, int D=3, int L=0)**
  Draw an 8-bit per color RGB or luminance image.

- **void fl_draw_image (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=3)**
  Draw an image using a callback function to generate image data.

- **void fl_draw_image_mono (const uchar *buf, int X, int Y, int W, int H, int D=1, int L=0)**
  Draw a gray-scale (1 channel) image.

- **void fl_draw_image_mono (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=1)**
  Draw a gray-scale image using a callback function to generate image data.

- **int fl_draw_pixmap (const char *const *data, int x, int y, Fl_Color bg=FL_GRAY)**
  Draw XPM image data, with the top-left corner at the given position.

- **int fl_draw_pixmap (char *const *data, int x, int y, Fl_Color bg=FL_GRAY)**
  Draw XPM image data, with the top-left corner at the given position.

- **void fl_draw_radio (int x, int y, int d, Fl_Color color)**
  Draw a round check mark (circle) of a radio button.

- **int fl_draw_symbol (const char *label, int x, int y, int w, int h, Fl_Color)**
  Draw the named symbol in the given rectangle using the given color.

- **void fl_end_complex_polygon ()**
  End complex filled polygon, and draw.

- **void fl_end_line ()**
  End list of lines, and draw.

- **void fl_end_loop ()**
  End closed sequence of lines, and draw.

- **void fl_end_offscreen ()**
  Quit sending drawing commands to the current offscreen buffer.

- **void fl_end_points ()**
  End list of points, and draw.

- **void fl_end_polygon ()**
  End convex filled polygon, and draw.

- **const char * fl_expand_text (const char *from, char *buf, int maxbuf, double maxw, int &n, double &width, int wrap, int draw_symbols=0)**
  Copy from to buf, replacing control characters with ^X.

- **void fl_focus_rect (int x, int y, int w, int h)**
  Draw a dotted rectangle, used to indicate keyboard focus on a widget.

- **void fl_frame (const char *s, int x, int y, int w, int h)**
  Draws a series of line segments around the given box.

- **void fl_frame2 (const char *s, int x, int y, int w, int h)**
  Draws a series of line segments around the given box.

- **void fl_gap ()**
  Separate loops of the path.

- **void fl_line (int x, int y, int x1, int y1)**
  Draw a line from (x,y) to (x1,y1)

- **void fl_line (int x, int y, int x1, int y1, int x2, int y2)**
  Draw a line from (x,y) to (x1,y1) and another from (x1,y1) to (x2,y2)

- **void fl_line_style (int style, int width=0, char *dashes=0)**
  Set how to draw lines (the “pen”).

- **void fl_load_identity ()**
  Set the transformation matrix to identity.

- **void fl_load_matrix (double a, double b, double c, double d, double x, double y)**
11.7 Drawing functions

Set the current transformation matrix.

- void fl_loop (int x, int y, int x1, int y1, int x2, int y2)
  Outline a 3-sided polygon with lines.

- void fl_loop (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)
  Outline a 4-sided polygon with lines.

- void fl_measure (const char *str, int &x, int &y, int draw_symbols=1)
  Measure how wide and tall the string will be when printed by the fl_draw() function with align parameter.

- int fl_measure_pixmap (char *const *data, int &w, int &h)
  Get the dimensions of a pixmap.

- int fl_measure_pixmap (const char *const *cdata, int &w, int &h)
  Get the dimensions of a pixmap.

- void fl_mult_matrix (double a, double b, double c, double d, double x, double y)
  Concatenate another transformation onto the current one.

- int fl_not_clipped (int x, int y, int w, int h)
  Does the rectangle intersect the current clip region?

- unsigned int fl_old_shortcut (const char *s)
  Emulation of XForms named shortcuts.

- void fl_overlay_clear ()
  Erase a selection rectangle without drawing a new one.

- void fl_overlay_rect (int x, int y, int w, int h)
  Draw a transient dotted selection rectangle.

- float fl_override_scale ()
  Removes any GUI scaling factor in subsequent drawing operations.

- void fl_pie (int x, int y, int w, int h, double a1, double a2)
  Draw filled ellipse sections using integer coordinates.

- void fl_point (int x, int y)
  Draw a single pixel at the given coordinates.

- void fl_polygon (int x, int y, int x1, int y1, int x2, int y2)
  Fill a 3-sided polygon.

- void fl_polygon (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)
  Fill a 4-sided polygon.

- void fl_pop_clip ()
  Restore the previous clip region.

- void fl_pop_matrix ()
  Restore the current transformation matrix from the stack.

- void fl_push_clip (int x, int y, int w, int h)
  Intersect the current clip region with a rectangle and push this new region onto the stack.

- void fl_push_matrix ()
  Save the current transformation matrix on the stack.

- void fl_push_no_clip ()
  Push an empty clip region onto the stack so nothing will be clipped.

- uchar * fl_read_image (uchar *p, int X, int Y, int W, int H, int alpha=0)
  Reads an RGB(A) image from the current window or off-screen buffer.

- void fl_rect (Fl_Rect r)
  Draw a border inside the given bounding box.

- void fl_rect (int x, int y, int w, int h)
  Draw a border inside the given bounding box.

- void fl_rect (int x, int y, int w, int h, Fl_Color c)
  Draw with passed color a border inside the given bounding box.

- void fl_rectf (Fl_Rect bb, uchar r, uchar g, uchar b)
  Color a rectangle with "exactly" the passed r, g, b color.
• void **fl_rectf** (Fl_Rect r)
  Color with current color a rectangle that exactly fills the given bounding box.
• void **fl_rectf** (Fl_Rect r, Fl_Color c)
  Color with passed color a rectangle that exactly fills the given bounding box.
• void **fl_rectf** (int x, int y, int w, int h)
  Color with current color a rectangle that exactly fills the given bounding box.
• void **fl_rectf** (int x, int y, int w, int h, Fl_Color c)
  Color with passed color a rectangle that exactly fills the given bounding box.
• void **fl_rectf** (int x, int y, int w, int h, uchar r, uchar g, uchar b)
  Color a rectangle with "exactly" the passed r, g, b color.
• void **fl_rescale_offscreen** (Fl_Offscreen &ctx)
  Adapts an offscreen buffer to a changed value of the scale factor.
• void **fl_reset_spot** (void)
  Resets marked text.
• void **fl_restore_clip** ()
  Undo any clobbering of the clip region done by your program.
• void **fl_restore_scale** (float s)
  Restores the GUI scaling factor and the clipping region in subsequent drawing operations.
• void **fl_rotate** (double d)
  Concatenate rotation transformation onto the current one.
• void **fl_rounded_rect** (int x, int y, int w, int h, int r)
  Draw a rounded border inside the given bounding box.
• void **fl_rounded_rectf** (int x, int y, int w, int h, int r)
  Color with current color a rounded rectangle that exactly fills the given bounding box.
• void **fl_rtl_draw** (const char * str, int n, int x, int y)
  Draw a UTF-8 string of length n bytes right to left starting at the given x, y location.
• void **fl_scale** (double x)
  Concatenate scaling transformation onto the current one.
• void **fl_scale** (double x, double y)
  Concatenate scaling transformation onto the current one.
• void **fl_scroll** (int X, int Y, int W, int H, int dx, int dy, void(*draw_area)(void *, int, int, int), void *data)
  Scroll a rectangle and draw the newly exposed portions.
• void **fl_set_spot** (int font, int size, int X, int Y, int W, int H, Fl_Window *win=0)
  Inform text input methods about the current text insertion cursor.
• void **fl_set_status** (int X, int Y, int W, int H)
  Related to text input methods under X11.
• const char * **fl_shortcut_label** (unsigned int shortcut)
  Get a human-readable string from a shortcut value.
• const char * **fl_shortcut_label** (unsigned int shortcut, const char **eom)
  Get a human-readable string from a shortcut value.
• double **fl_transform_dx** (double x, double y)
  Transform distance using current transformation matrix.
• double **fl_transform_dy** (double x, double y)
  Transform distance using current transformation matrix.
• double **fl_transform_x** (double x, double y)
  Transform coordinate using the current transformation matrix.
• double **fl_transform_y** (double x, double y)
  Transform coordinate using the current transformation matrix.
• void **fl_transformed_vertex** (double xf, double yf)
  Add coordinate pair to the vertex list without further transformations.
• void **fl_translate** (double x, double y)
11.7 Drawing functions

Concatenate translation transformation onto the current one.

- void fl_vertex (double x, double y)
  
  Add a single vertex to the current path.

- void fl_xline (int x, int y, int x1)
  
  Draw a horizontal line from (x,y) to (x1,y).

- void fl_yxline (int x, int y, int x1, int y2)
  
  Draw a horizontal line from (x,y) to (x1,y), then vertical from (x1,y) to (x1,y2).

- void fl_yxline (int x, int y, int x1, int y2, int x3)
  
  Draw a horizontal line from (x,y) to (x1,y), then a vertical from (x1,y) to (x1,y2) and then another horizontal from (x1,y2) to (x3,y2).

- void fl_yxline (int x, int y, int y1)
  
  Draw a vertical line from (x,y) to (x,y1)

- void fl_yxline (int x, int y, int y1, int x2)
  
  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1).

- void fl_yxline (int x, int y, int y1, int x2, int y3)
  
  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1), then another vertical from (x2,y1) to (x2,y3).

11.7.1 Detailed Description

FLTK global graphics and GUI drawing functions. These functions are declared in `<FL/fl_draw.H>`, and in `<FL/platform.H>` for offscreen buffer-related ones.

11.7.2 Enumeration Type Documentation

11.7.2.1 anonymous enum

anonymous enum

enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_SOLID</td>
<td>line style: ____________</td>
</tr>
<tr>
<td>FL_DASH</td>
<td>line style: _ _ _ _ _ _ _ _</td>
</tr>
<tr>
<td>FL_DOT</td>
<td>line style: . . . . . .</td>
</tr>
<tr>
<td>FL_DASHDOT</td>
<td>line style: _ _ _ _ _ _</td>
</tr>
<tr>
<td>FL_DASHDOTDOT</td>
<td>line style: _ _ _ _ _ _ _ _ _ _</td>
</tr>
<tr>
<td>FL_CAP_FLAT</td>
<td>cap style: end is flat</td>
</tr>
<tr>
<td>FL_CAP_ROUND</td>
<td>cap style: end is round</td>
</tr>
<tr>
<td>FL_CAP_SQUARE</td>
<td>cap style: end wraps end point</td>
</tr>
<tr>
<td>FL_JOIN_MITER</td>
<td>join style: line join extends to a point</td>
</tr>
<tr>
<td>FL_JOIN_ROUND</td>
<td>join style: line join is rounded</td>
</tr>
<tr>
<td>FL_JOIN_BEVEL</td>
<td>join style: line join is tidied</td>
</tr>
</tbody>
</table>

11.7.3 Function Documentation

11.7.3.1 fl_add_symbol()

```c
int fl_add_symbol (  
  const char * name,  
  void(*)(Fl_Color) drawit,  
  int scalable )
```

Adds a symbol to the system.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>name of symbol (without the &quot;@&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>drawit</td>
<td>function to draw symbol</td>
</tr>
<tr>
<td>in</td>
<td>scalable</td>
<td>set to 1 if drawit uses scalable vector drawing</td>
</tr>
</tbody>
</table>

Returns

1 on success, 0 on failure

### 11.7.3.2 fl_antialias()

```c
void fl_antialias ( int state ) [inline]
```

Turn antialiased line drawings ON or OFF, if supported by platform. Currently, only the Windows platform allows to change whether line drawings are antialiased. Turning it OFF may accelerate heavy drawing operations.

### 11.7.3.3 fl_arc() [1/2]

```c
void fl_arc ( double x, double y, double r, double start, double end ) [inline]
```

Add a series of points to the current path on the arc of a circle. You can get elliptical paths by using scale and rotate before calling fl_arc().

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x,y,r</th>
<th>center and radius of circular arc</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>start,end</td>
<td>angles of start and end of arc measured in degrees counter-clockwise from 3 o'clock. If end is less than start then it draws the arc in a clockwise direction.</td>
</tr>
</tbody>
</table>

![Figure 11.2 fl_arc(x,y,r,a1,a2)](image_url)

Examples:

// Draw an arc of points
fl_begin_points();
fl_arc(100.0, 100.0, 50.0, 0.0, 180.0);
fl_end_points();

// Draw arc with a line
11.7 Drawing functions

11.7.3.4 \texttt{fl\_arc()} [2/2]

```c
void fl\_arc (int \_x, int \_y, int \_w, int \_h, double \_a1, double \_a2) [inline]
```

Draw ellipse sections using integer coordinates. These functions match the rather limited circle drawing code provided by X and Windows. The advantage over using \texttt{fl\_arc} with floating point coordinates is that they are faster because they often use the hardware, and they draw much nicer small circles, since the small sizes are often hard-coded bitmaps.

If a complete circle is drawn it will fit inside the passed bounding box. The two angles are measured in degrees counter-clockwise from 3 o'clock and are the starting and ending angle of the arc, \( \_a2 \) must be greater or equal to \( \_a1 \).

\texttt{fl\_arc()} draws a series of lines to approximate the arc. Notice that the integer version of \texttt{fl\_arc()} has a different number of arguments than the double version \texttt{fl\_arc(double \_x, double \_y, double \_r, double \_start, double \_end)}.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{x,y,w,h}</td>
<td>bounding box of complete circle</td>
</tr>
<tr>
<td>\texttt{a1,a2}</td>
<td>start and end angles of arc measured in degrees counter-clockwise from 3 o'clock. ( _a2 ) must be greater than or equal to ( _a1 ).</td>
</tr>
</tbody>
</table>

11.7.3.5 \texttt{fl\_begin\_complex\_polygon()}

```c
void fl\_begin\_complex\_polygon ( ) [inline]
```

Start drawing a complex filled polygon. The polygon may be concave, may have holes in it, or may be several disconnected pieces. Call \texttt{fl\_gap()} to separate loops of the path.

To outline the polygon, use \texttt{fl\_begin\_loop()} and replace each \texttt{fl\_gap()} with \texttt{fl\_end\_loop();fl\_begin\_loop()} pairs.

Note

For portability, you should only draw polygons that appear the same whether "even/odd" or "non-zero" winding rules are used to fill them. Holes should be drawn in the opposite direction to the outside loop.

Figure 11.3 \texttt{fl\_pie()} and \texttt{fl\_arc()}

Generated by Doxygen
11.7.3.6  fl_begin_offscreen()

void fl_begin_offscreen (  
                   Fl_Offscreen  ctx  )  
Send all subsequent drawing commands to this offscreen buffer.

Parameters

ctx  the offscreen buffer.

Note

The ctx argument must have been created by fl_create_offscreen().

11.7.3.7  fl_begin_points()

void fl_begin_points ( )  [inline]  
Start drawing a list of points.
Points are added to the list with fl_vertex().

11.7.3.8  fl_can_do_alpha_blending()

cchar fl_can_do_alpha_blending ( )  [inline]  
Check whether platform supports true alpha blending for RGBA images.

Returns

1 if true alpha blending supported by platform
0 not supported so FLTK will use screen door transparency

11.7.3.9  fl_capture_window()

Fl_RGB_Image ∗ fl_capture_window (  
                   Fl_Window ∗  win,  
                   int  x,  
                   int  y,  
                   int  w,  
                   int  h  )  
Captures the content of a rectangular zone of a mapped window.

Parameters

<table>
<thead>
<tr>
<th>win</th>
<th>a mapped Fl_Window (derived types including Fl_Gl_Window are also possible)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x,y,w,h</td>
<td>window area to be captured. Intersecting sub-windows are captured too.</td>
</tr>
</tbody>
</table>

Returns

The captured pixels as an Fl_RGB_Image. The raw and drawing sizes of the image can differ. Returns NULL when capture was not successful. The image depth may differ between platforms.

Version

1.4

11.7.3.10  fl_circle()

void fl_circle (  
                   double  x,  
                   }
}
double y,
    double r } [inline]

fl_circle(x,y,r) is equivalent to fl_arc(x,y,r,0,360), but may be faster.

Parameters

| in | x,y,r | center and radius of circle |

Note

fl_circle() is best used as part of the Drawing Complex Shapes API, that is, flanked by fl_begin_XXX() and fl_end_XXX() calls where XXX can be 'loop' or 'polygon' to draw, respectively a circle or a disk. Transformation functions (e.g., fl_scale(double, double)) can be also used for fl_circle() to draw empty of filled ellipses. It must be the only thing in the path: if you want a circle as part of a complex polygon you must use fl_arc(). Nevertheless, fl_circle() can also be used by itself to draw circles.

11.7.3.11 fl_clip()

void fl_clip {
    int x,
    int y,
    int w,
    int h ) [inline]

Intersect the current clip region with a rectangle and push this new region onto the stack (deprecated).

Parameters

| in | x,y,w,h | position and size |

Deprecated Please use fl_push_clip(int x, int y, int w, int h) instead. fl_clip(int, int, int, int) will be removed in FLTK 1.5.

11.7.3.12 fl_clip_box()

int fl_clip_box (   int x,
    int y,
    int w,
    int h,
    int & X,
    int & Y,
    int & W,
    int & H ) [inline]

Intersect a rectangle with the current clip region and return the bounding box of the result.

Returns non-zero if the resulting rectangle is different to the original. The given rectangle (x, y, w, h) should be entirely inside its window, otherwise the result may be unexpected, i.e. this function may not clip the rectangle to the window coordinates and size. In particular x and y should not be negative.

The resulting bounding box can be used to limit the necessary drawing to this rectangle.

Example:

```cpp
void MyGroup::draw() {
    int X = 0, Y = 0, W = 0, H = 0;
    int ret = fl_clip_box(x(), y(), w(), h(), X, Y, W, H);
    if (ret == 0) { // entire group is visible (not clipped)
        // full drawing code here
    } else { // parts of this group are clipped
        // partial drawing code here (uses X, Y, W, and H to test)
    }
}
```
and \( h \) are set to zero if the rectangle is completely outside the clipping region. In this case \( x \) and \( y \) are undefined and should not be used. Possible values are \( (0, 0) \), \( (x, y) \), or anything else (platform dependent).

**Note**

This function is platform-dependent. If the given rectangle is not entirely inside the window, the results are not guaranteed to be the same on all platforms.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>x, y, w, h</th>
<th>position and size of rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>X, Y, W, H</td>
<td>position and size of resulting bounding box.</td>
</tr>
</tbody>
</table>

**Returns**

Non-zero if the resulting rectangle is different to the original.

**See also**

fl_not_clipped()

### 11.7.3.13 fl_clip_region() [1/2]

**FlRegion** fl_clip_region () [inline]

Return the current clipping region.

**Note**

This function is mostly intended for internal use by the FLTK library when drawing to the display. Its return value can be always NULL if the current drawing surface is not the display.

### 11.7.3.14 fl_clip_region() [2/2]

void fl_clip_region (FlRegion r) [inline]

Replace the top of the clipping stack with a clipping region of any shape. FlRegion is an operating system specific type.

**Note**

This function is mostly intended for internal use by the FLTK library when drawing to the display. Its effect can be null if the current drawing surface is not the display.

**Parameters**

| in | r | clipping region |

### 11.7.3.15 fl_copy_offscreen()

void fl_copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx,
11.7 Drawing functions

Copy a rectangular area of the given offscreen buffer into the current drawing destination.

Parameters

<table>
<thead>
<tr>
<th>x,y</th>
<th>position where to draw the copied rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>w,h</td>
<td>size of the copied rectangle</td>
</tr>
<tr>
<td>pixmap</td>
<td>offscreen buffer containing the rectangle to copy</td>
</tr>
<tr>
<td>srcx,srcy</td>
<td>origin in offscreen buffer of rectangle to copy</td>
</tr>
</tbody>
</table>

### 11.7.3.16 fl_create_offscreen()

```c
Fl_Offscreen fl_create_offscreen (  
  int w,  
  int h )
```

Creation of an offscreen graphics buffer.

Parameters

| w,h  | width and height in FLTK units of the buffer. |

Returns

the created graphics buffer.

The pixel size of the created graphics buffer is equal to the number of pixels in an area of the screen containing the current window sized at w,h FLTK units. This pixel size varies with the value of the scale factor of this screen.

Note

Work with the fl_XXX_offscreen() functions is equivalent to work with an Fl_Image_Surface object, as follows:

<table>
<thead>
<tr>
<th>Fl_Offscreen-based approach</th>
<th>Fl_Image_Surface-based approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Offscreen off = fl_create_offscreen(w, h)</td>
<td>Fl_Image_Surface *surface = new Fl_Image_Surface(w, h, 1)</td>
</tr>
<tr>
<td>fl_begin_offscreen(off)</td>
<td>Fl_Surface_Device::push_current(surface)</td>
</tr>
<tr>
<td>fl_end_offscreen()</td>
<td>Fl_Surface_Device::pop_current()</td>
</tr>
<tr>
<td>fl_copy_offscreen(x,y,w,h, off, sx,sy)</td>
<td>fl_copy_offscreen(x,y,w,h, surface-&gt;offscreen(), sx,sy)</td>
</tr>
<tr>
<td>fl_rescale_offscreen(off)</td>
<td>surface-&gt;rescale()</td>
</tr>
<tr>
<td>fl_delete_offscreen(off)</td>
<td>delete surface</td>
</tr>
</tbody>
</table>

### 11.7.3.17 fl_cursor()

```c
void fl_cursor (  
  Fl_Cursor c )
```

Sets the cursor for the current window to the specified shape and colors.

The cursors are defined in the `<FL/Enumerations.H>` header file.

### 11.7.3.18 fl_curve()

```c
void fl_curve (  
  double X0,  
  double Y0,  
  double X1,  
  double Y1,  
  ...  
)```
double X2,
double Y2,
double X3,
double Y3) [inline]

Add a series of points on a Bézier curve to the path.
The curve ends (and two of the points) are at X0,Y0 and X3,Y3.

### Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X0,Y0</td>
<td>curve start point</td>
</tr>
<tr>
<td>X1,Y1</td>
<td>curve control point</td>
</tr>
<tr>
<td>X2,Y2</td>
<td>curve control point</td>
</tr>
<tr>
<td>X3,Y3</td>
<td>curve end point</td>
</tr>
</tbody>
</table>

#### 11.7.3.19 fl_delete_offscreen()

```c
void fl_delete_offscreen (  
  Fl_Offscreen ctx  )
```

Deletion of an offscreen graphics buffer.

**Parameters**

| ctx    | the buffer to be deleted. |

**Note**

The `ctx` argument must have been created by `fl_create_offscreen()`.

#### 11.7.3.20 fl_draw() [1/4]

```c
void fl_draw (  
  const char * str,  
  int x,  
  int y  )
```

Draw a null-terminated UTF-8 string starting at the given `x`, `y` location.

Text is aligned to the left and to the baseline of the font. To align to the bottom, subtract `fl_descent()` from `y`. To align to the top, subtract `fl_descent()` and add `fl_height()`. This version of `fl_draw` provides direct access to the text drawing function of the underlying OS. It does not apply any special handling to control characters.

#### 11.7.3.21 fl_draw() [2/4]

```c
void fl_draw (  
  const char * str,  
  int x,  
  int y,  
  int w,  
  int h,  
  Fl_Align align,  
  Fl_Image * img,  
  int draw_symbols  )
```

Fancy string drawing function which is used to draw all the labels.

The string is formatted and aligned inside the passed box. Handles 't' and 'n', expands all other control characters to '^X', and aligns inside or against the edges of the box. See `Fl_Widget::align()` for values of `align`. The value `FL_ALIGN_INSIDE` is ignored, as this function always prints inside the box. If `img` is provided and is not `NULL`, the image is drawn above or below the text as specified by the `align` value. The `draw_symbols` argument specifies whether or not to look for symbol names starting with the '@' character.
11.7 Drawing functions

11.7.3.22 fl_draw() [3/4]

```c
void fl_draw (  
    int angle,  
    const char * str,  
    int n,  
    int x,  
    int y ) [inline]
```

Draw at the given x, y location a UTF-8 string of length n bytes rotating angle degrees counter-clockwise.

**Note**

When using X11 (Unix, Linux, Cygwin et al.) this needs Xft to work. Under plain X11 (w/o Xft) rotated text is not supported by FLTK. A warning will be issued to stderr at runtime (only once) if you use this method with an angle other than 0.

11.7.3.23 fl_draw() [4/4]

```c
void fl_draw (  
    int angle,  
    const char * str,  
    int x,  
    int y)
```

Draw a nul-terminated UTF-8 string starting at the given x, y location and rotating angle degrees counter-clockwise.

This version of fl_draw provides direct access to the text drawing function of the underlying OS and is supported by all fltk platforms except X11 without Xft.

11.7.3.24 fl_draw_arrow()

```c
void fl_draw_arrow (  
    Fl_Rect r,  
    Fl_Arrow_Type t,  
    Fl_Orientation o,  
    Fl_Color col )
```

Draw an "arrow like" GUI element for the selected scheme.

In the future this function should be integrated in Fl_Scheme as a virtual method, i.e. it would call a method like ...

```c
Fl_Scheme::current()->draw_arrow(r, t, o, col);
```

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>r</th>
<th>bounding box</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>t</td>
<td>arrow type</td>
</tr>
<tr>
<td>in</td>
<td>o</td>
<td>orientation</td>
</tr>
<tr>
<td>in</td>
<td>col</td>
<td>arrow color</td>
</tr>
</tbody>
</table>

Since

1.4.0

11.7.3.25 fl_draw_box()

```c
void fl_draw_box (  
    Fl_Boxtype t,  
    int x,  
    int y,  
    int w,  
    int h,  
    Fl_Color c )
```
Draws a box using given type, position, size and color.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>box type</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y,w,h</td>
<td>position and size</td>
</tr>
<tr>
<td>in</td>
<td>c</td>
<td>color</td>
</tr>
</tbody>
</table>

### 11.7.3.26 fl_draw_check()

```c
void fl_draw_check (
    Fl_Rect bb,
    Fl_Color col )
```

Draw a check mark inside the given bounding box.

The check mark is allowed to fill the entire box but the algorithm used makes sure that a 1-pixel border is kept free if the box is large enough. You need to calculate margins for box borders etc. yourself.

The check mark size is limited (minimum and maximum size) and the check mark is always centered in the given box.

**Note**

If the box is too small (bad GUI design) the check mark will be drawn over the box borders. This is intentional for better user experience. Otherwise users might not be able to recognize if a box is checked.

The size limits are implementation details and may be changed at any time.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>bb</th>
<th>rectangle that defines the bounding box</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>col</td>
<td>Fl_Color to draw the check mark</td>
</tr>
</tbody>
</table>

Since

1.4.0

### 11.7.3.27 fl_draw_circle()

```c
void fl_draw_circle ( 
    int x, 
    int y, 
    int d, 
    Fl_Color color )
```

Draw a potentially small, filled circle using a given color.

This function draws a filled circle bounded by rectangle (x, y, d, d) using color color

This function is the same as fl_pie(x, y, d, d, 0, 360) except with some systems that don't draw small circles well. In that situation, the circle diameter d is converted from FLTK units to pixels and this function approximates a filled circle by drawing several filled rectangles if the converted diameter is 6 pixels.

The current drawing color fl_color() is preserved across the call.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>x,y</th>
<th>coordinates of top left of the bounding box</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>d</td>
<td>diameter == width and height of the bounding box in FLTK units</td>
</tr>
<tr>
<td>in</td>
<td>color</td>
<td>the color used to draw the circle</td>
</tr>
</tbody>
</table>
11.7 Drawing functions

Since 1.4.0

11.7.3.28 fl_draw_image() [1/2]

void fl_draw_image (  
    const uchar * buf,  
    int X,  
    int Y,  
    int W,  
    int H,  
    int D = 3,  
    int L = 0 ) [inline]

Draw an 8-bit per color RGB or luminance image.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>buf</th>
<th>points at the &quot;r&quot; data of the top-left pixel. Color data must be in r,g,b order. Luminance data is only one gray byte.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y</td>
<td>position where to put top-left corner of image</td>
</tr>
<tr>
<td>in</td>
<td>W,H</td>
<td>size of the image</td>
</tr>
<tr>
<td>in</td>
<td>D</td>
<td>delta to add to the pointer between pixels. It may be any value greater than or equal to 1, or it can be negative to flip the image horizontally</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
<td>delta to add to the pointer between lines (if 0 is passed it uses W * D), and may be larger than W * D to crop data, or negative to flip the image vertically</td>
</tr>
</tbody>
</table>

It is highly recommended that you put the following code before the first show() of any window in your program to get rid of the dithering if possible:

```c
Fl::visual(Fl_RGB);
```

Gray scale (1-channel) images may be drawn. This is done if abs(D) is less than 3, or by calling fl_draw_image_mono(). Only one 8-bit sample is used for each pixel, and on screens with different numbers of bits for red, green, and blue only gray colors are used. Setting D greater than 1 will let you display one channel of a color image.

Note:

The X version does not support all possible visuals. If FLTK cannot draw the image in the current visual it will abort. FLTK supports any visual of 8 bits or less, and all common TrueColor visuals up to 32 bits.

11.7.3.29 fl_draw_image() [2/2]

void fl_draw_image (  
    Fl_Draw_Image_Cb cb,  
    void * data,  
    int X,  
    int Y,  
    int W,  
    int H,  
    int D = 3 ) [inline]

Draw an image using a callback function to generate image data.

You can generate the image as it is being drawn, or do arbitrary decompression of stored data, provided it can be decompressed to individual scan lines.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cb</th>
<th>callback function to generate scan line data</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>user data passed to callback function</td>
</tr>
</tbody>
</table>
Parameters

| in | X,Y | screen position of top left pixel |
| in | W,H | image width and height |
| in | D  | data size per pixel in bytes (must be greater than 0) |

See also

`fl_draw_image(const uchar * buf, int X, int Y, int W, int H, int D, int L)`

The callback function `cb` is called with the `void *data` user data pointer to allow access to a structure of information about the image, and the x, y, and w of the scan line desired from the image. 0,0 is the upper-left corner of the image, not x, y. A pointer to a buffer to put the data into is passed. You must copy w pixels from scanline y, starting at pixel x, to this buffer.

Due to cropping, less than the whole image may be requested. So x may be greater than zero, the first y may be greater than zero, and w may be less than W. The buffer is long enough to store the entire W * D pixels, this is for convenience with some decompression schemes where you must decompress the entire line at once: decompress it into the buffer, and then if x is not zero, copy the data over so the x'th pixel is at the start of the buffer.

You can assume the y's will be consecutive, except the first one may be greater than zero. If D is 4 or more, you must fill in the unused bytes with zero.

### 11.7.3.30 fl_draw_image_mono() [1/2]

```c
void fl_draw_image_mono ( 
    const uchar * buf, 
    int X, 
    int Y, 
    int W, 
    int H, 
    int D = 1, 
    int L = 0 ) [inline]
```

Draw a gray-scale (1 channel) image.

See also

`fl_draw_image(const uchar * buf, int X, int Y, int W, int H, int D, int L)`

### 11.7.3.31 fl_draw_image_mono() [2/2]

```c
void fl_draw_image_mono ( 
    Fl_Draw_Image_Cb cb, 
    void * data, 
    int X, 
    int Y, 
    int W, 
    int H, 
    int D = 1 ) [inline]
```

Draw a gray-scale image using a callback function to generate image data.

See also

`fl_draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D)`

### 11.7.3.32 fl_draw_pixmap() [1/2]

```c
int fl_draw_pixmap ( 
    char *const * data, 
    int x, 
    int y, 
    Fl_Color bg = FL_GRAY ) [inline]
```

Draw XPM image data, with the top-left corner at the given position.
11.7 Drawing functions

See also

fl_draw_pixmap(const char* const* data, int x, int y, Fl_Color bg)

11.7.3.33 fl_draw_pixmap() [2/2]

int fl_draw_pixmap (  
    const char *const * data,  
    int x,  
    int y,  
    Fl_Color bg = FL_GRAY  )

Draw XPM image data, with the top-left corner at the given position.  
The image is dithered on 8-bit displays so you won't lose color space for programs displaying both images and pixmaps.

Parameters

| in  | data          | pointer to XPM image data |
| in  | x,y           | position of top-left corner |
| in  | bg            | background color |

Returns

0 if there was any error decoding the XPM data.

11.7.3.34 fl_draw_radio()

void fl_draw_radio (  
    int x,  
    int y,  
    int d,  
    Fl_Color color  )

Draw a round check mark (circle) of a radio button.  
This draws only the round "radio button mark", it does not draw the (also typically round) box of the radio button.  
Call this only if the radio button is ON.  
This method draws a scheme specific "circle" with a particular light effect if the scheme is gtk+. For all other schemes this function draws a simple, small circle.  
The color must be chosen by the caller so it has enough contrast with the background.  
The bounding box of the circle is the rectangle (x, y, d, d).  
The current drawing color fl_color() is preserved across the call.

Parameters

| in  | x,y   | coordinates of top left of the bounding box |
| in  | d     | diameter == width and height of the bounding box in FLTK units |
| in  | color | the base color used to draw the circle |

Since

1.4.0

11.7.3.35 fl_draw_symbol()

int fl_draw_symbol (  
    const char * label,  
    int x,  

Generated by Doxygen
int y,
int w,
int h,
Fl_Color col
)
Draw the named symbol in the given rectangle using the given color.

Parameters

<table>
<thead>
<tr>
<th>in label</th>
<th>name of symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>in x,y</td>
<td>position of symbol</td>
</tr>
<tr>
<td>in w,h</td>
<td>size of symbol</td>
</tr>
<tr>
<td>in col</td>
<td>color of symbox</td>
</tr>
</tbody>
</table>

Returns

1 on success, 0 on failure

11.7.3.36 fl_expand_text()

const char * fl_expand_text (  
    const char * from,
    char * buf,
    int maxbuf,
    double maxw,
    int & n,
    double & width,
    int wrap,
    int draw_symbols )

Copy from to buf, replacing control characters with ^X. Stop at a newline or if maxbuf characters written to buffer. Also word-wrap if width exceeds maxw. Returns a pointer to the start of the next line of characters. Sets n to the number of characters put into the buffer. Sets width to the width of the string in the current font.

11.7.3.37 fl_focus_rect()

void fl_focus_rect (  
    int x,
    int y,
    int w,
    int h ) [inline]

Draw a dotted rectangle, used to indicate keyboard focus on a widget. This method draws the rectangle in the current color and independent of the Fl::visible_focus() option. You may need to set the current color with fl_color() before you call this.

11.7.3.38 fl_frame()

void fl_frame (  
    const char * s,
    int x,
    int y,
    int w,
    int h )

Draws a series of line segments around the given box. The string s must contain groups of 4 letters which specify one of 24 standard grayscale values, where 'A' is black and 'X' is white. The order of each set of 4 characters is: top, left, bottom, right. The result of calling fl_frame() with a string that is not a multiple of 4 characters in length is undefined. The only difference between this function and fl_frame2() is the order of the line segments.
11.7 Drawing functions

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>s</td>
</tr>
<tr>
<td>in</td>
<td>x,y,w,h</td>
</tr>
</tbody>
</table>

11.7.3.39 fl_frame2()

```c
void fl_frame2 ( const char * s, int x, int y, int w, int h )
```

Draws a series of line segments around the given box. The string `s` must contain groups of 4 letters which specify one of 24 standard grayscale values, where 'A' is black and 'X' is white. The order of each set of 4 characters is: bottom, right, top, left. The result of calling `fl_frame2()` with a string that is not a multiple of 4 characters in length is undefined. The only difference between this function and `fl_frame()` is the order of the line segments.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>s</td>
</tr>
<tr>
<td>in</td>
<td>x,y,w,h</td>
</tr>
</tbody>
</table>

11.7.3.40 fl_gap()

```c
void fl_gap ( ) [inline]
```

Separate loops of the path. It is unnecessary but harmless to call `fl_gap()` before the first vertex, after the last vertex, or several times in a row.

11.7.3.41 fl_line_style()

```c
void fl_line_style ( int style, int width = 0, char * dashes = 0 ) [inline]
```

Set how to draw lines (the "pen"). If you change this it is your responsibility to set it back to the default using `fl_line_style(0)`.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>style</td>
</tr>
<tr>
<td>in</td>
<td>width</td>
</tr>
<tr>
<td>in</td>
<td>dashes</td>
</tr>
</tbody>
</table>
Note
Because of how line styles are implemented on Win32 systems, you must set the line style after setting the
drawing color. If you set the color after the line style you will lose the line style settings.
The dashes array does not work under the (unsupported!) operating systems Windows 95, 98 or Me, since
those operating systems do not support complex line styles.

11.7.3.42 fl_load_matrix()

void fl_load_matrix (  
  double a,  
  double b,  
  double c,  
  double d,  
  double x,  
  double y ) [inline]

Set the current transformation matrix.

Parameters

| in | a, b, c, d, x, y | transformation matrix elements |

11.7.3.43 fl_measure()

void fl_measure (  
  const char * str,  
  int & w,  
  int & h,  
  int draw_symbols )

Measure how wide and tall the string will be when printed by the fl_draw() function with align parameter.
If the incoming w is non-zero it will wrap to that width.
The current font is used to do the width/height calculations, so unless its value is known at the time fl_measure() is
called, it is advised to first set the current font with fl_font(). With event-driven GUI programming you can never be
sure which widget was exposed and redrawn last, nor which font it used. If you have not called fl_font() explicitly in
your own code, the width and height may be set to unexpected values, even zero!
Note: In the general use case, it’s a common error to forget to set w to 0 before calling fl_measure() when wrap
behavior isn’t needed.

Parameters

| in | str | nul-terminated string |
| out | w, h | width and height of string in current font |
| in | draw_symbols | non-zero to enable @symbol handling [default=1] |

// Example: Common use case for fl_measure()
const char *s = "This is a test";  // initialize to zero before calling fl_measure()
fl_font(FL_HELVETICA, 14);  // set current font face/size to be used for measuring
fl_measure(s, w, h);  // returns pixel width/height of string in current font

11.7.3.44 fl_measure_pixmap() [1/2]

int fl_measure_pixmap (  
  char *const * data,  
  int & w,  
  int & h )

Get the dimensions of a pixmap.
An XPM image contains the dimensions in its data. This function returns the width and height.
11.7 Drawing functions

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>data</th>
<th>pointer to XPM image data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>w,h</td>
<td>width and height of image</td>
</tr>
</tbody>
</table>

Returns

- non-zero if the dimensions were parsed OK
- 0 if there were any problems

11.7.3.45 fl_measure_pixmap() [2/2]

```c
int fl_measure_pixmap (const char * const * cdata, int & w, int & h )
```

Get the dimensions of a pixmap.

See also

fl_measure_pixmap(char * const * data, int &w, int &h)

11.7.3.46 fl_mult_matrix()

```c
void fl_mult_matrix (double a, double b, double c, double d, double x, double y ) [inline]
```

Concatenate another transformation onto the current one.

Parameters

| in | a,b,c,d,x,y     | transformation matrix elements such that \( X' = aX + cY + x \) and \( Y' = bX + dY + y \) |

11.7.3.47 fl_not_clipped()

```c
int fl_not_clipped (int x, int y, int w, int h ) [inline]
```

Does the rectangle intersect the current clip region?

Parameters

| in | x,y,w,h     | position and size of rectangle |

Returns

- non-zero if any of the rectangle intersects the current clip region. If this returns 0 you don't have to draw the object.
328 Topic Documentation

Note

Under X this returns 2 if the rectangle is partially clipped and 1 if it is entirely inside the clip region.

See also

fl_clip_box()

11.7.3.48 fl_oldShortcut()

unsigned int fl_oldShortcut (const char * s )

Emulation of XForms named shortcuts.

Converts ASCII shortcut specifications (eg. "^c") into the FLTK integer equivalent (eg. FL_CTRL+’c’)

These ASCII characters are used to specify the various keyboard modifier keys:

- # - Alt
- + - Shift
- ^ - Control
- ! - Meta
- @ - Command (Ctrl on linux/win, Meta on OSX)

These special characters can be combined to form chords of modifier keys. (See 'Remarks' below)

After the optional modifier key prefixes listed above, one can either specify a single keyboard character to use as the shortcut, or a numeric sequence in hex, decimal or octal.

Examples:

- "c" -- Uses ‘c’ as the shortcut
- "#c" -- Same as FL_ALT|FL_CTRL+’c’
- "@c" -- Same as FL_COMMAND|’c’ (see FL_COMMAND for platform specific behavior)
- "0x63" -- Same as "c" (hex 63==’c’)
- "99" -- Same as "c" (dec 99==’c’)
- "0143" -- Same as "c" (octal 0143==’c’)
- "^0x63" -- Same as (FL_CTRL|’c’), or (FL_CTRL|0x63)
- "^99" -- Same as (FL_CTRL|’c’), or (FL_CTRL|99)
- "^0143" -- Same as (FL_CTRL|’c’), or (FL_CTRL|0143)

Remarks

Due to XForms legacy, there are some odd things to consider when using the modifier characters.

(1) You can use the special modifier keys for chords only if the modifiers are provided in this order: #, +, ^, !, @. Other ordering can yield undefined results.

So for instance, Ctrl-Alt-c must be specified as "#^c" (and not "^#c"), due to the above ordering rule.

(2) If you want to make a shortcut that uses one of the special modifier characters (as the character being modified), then to avoid confusion, specify the numeric equivalent, e.g.

<table>
<thead>
<tr>
<th>If you want..</th>
<th>Then use..</th>
</tr>
</thead>
<tbody>
<tr>
<td>’#’ as the shortcut..</td>
<td>&quot;0x23&quot; (instead of just &quot;#&quot;)</td>
</tr>
<tr>
<td>’+’ as the shortcut..</td>
<td>&quot;0x2b&quot; (instead of just &quot;+&quot;)</td>
</tr>
<tr>
<td>’^’ as the shortcut..</td>
<td>&quot;0x5e&quot; (instead of just &quot;^&quot;)</td>
</tr>
<tr>
<td>Alt-+ as the shortcut..</td>
<td>&quot;#0x2b&quot; (instead of &quot;+&quot;)</td>
</tr>
<tr>
<td>Alt-^ as the shortcut..</td>
<td>&quot;#0x5e&quot; (instead of &quot;^&quot;)</td>
</tr>
</tbody>
</table>

As a general rule that's easy to remember, unless the shortcut key to be modified is a single alpha-numeric character [A-Z,a-z,0-9], it's probably best to use the numeric equivalents.

Don't fix these silly legacy issues in a future release. Nobody is using this anymore.
11.7.3.49 fl_overlay_clear()

void fl_overlay_clear ( )
Erase a selection rectangle without drawing a new one.

See also

fl_overlay_rect(int x, int y, int w, int h)

11.7.3.50 fl_overlay_rect()

void fl_overlay_rect ( )
  int x,
  int y,
  int w,
  int h )

Draw a transient dotted selection rectangle.
This function saves the current screen content and then draws a dotted selection rectangle into the front screen
buffer. If another selection rectangle was drawn earlier, the previous screen graphics are restored first.
To clear the selection rectangle, call fl_overlay_clear().
The typical (and only) use for this function is to draw a selection rectangle during a mouse drag event sequence
without having to redraw the entire content of the widget.

Your event handle should look similar to this (also see test/mandelbrot.cxx):

int MyWidget::handle(int event) {
  static int ix, iy;
  switch (event) {
    case FL_PUSH:
      ix = Fl::event_x(); iy = Fl::event_y();
      return 1;
    case FL_DRAG:
      this->make_current();
      fl_overlay_rect(ix, iy, ix-Fl::event_x(), iy-Fl::event_y());
      return 1;
    case FL_RELEASE:
      this->make_current();
      fl_overlay_clear();
      // select the element under the rectangle
      return 1;
    }
  return MySuperWidget::handle(event);
}

Note

Between drawing an overlay rect and clearing it, the content of the widget must not change.

fl_overlay_rect() and fl_overlay_clear() should be called when the actual event occurs, and not within My-
Widget::draw().

fl_overlay_rect() and fl_overlay_clear() should not be mixed with Fl_Overlay_Window. Fl_Overlay_Window
provides an entirely different way of drawing selection outlines and is not limited to rectangles.

Parameters

x,y,w,h  position and size of the overlay rectangle.

See also

fl_overlay_clear()

11.7.3.51 fl_override_scale()

float fl_override_scale ( )

Removes any GUI scaling factor in subsequent drawing operations.
This must be matched by a later call to fl_restore_scale(). This function can be used to transiently perform drawing
operations that are not rescaled by the current value of the GUI scaling factor. The resulting drawing context has no
clipping region.

Returns

The GUI scaling factor value that was in place when the function started.

### 11.7.3.52 \texttt{fl\_pie()}

```c
void fl_pie (  
    int x,  
    int y,  
    int w,  
    int h,  
    double a1,  
    double a2 ) [inline]
```

Draw filled ellipse sections using integer coordinates.

Like \texttt{fl\_arc()}, but \texttt{fl\_pie()} draws a filled-in pie slice. This slice may extend outside the line drawn by \texttt{fl\_arc()}; to avoid this use \texttt{w - 1} and \texttt{h - 1}.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x,y,w,h</th>
<th>bounding box of complete circle</th>
</tr>
</thead>
</table>
| in          | a1,a2  | start and end angles of arc measured in degrees counter-clockwise from 3 o'clock. \( a_2 \) must be greater than or equal to \( a_1 \).

![Figure 11.4 fl\_pie() and fl\_arc()](image)

### 11.7.3.53 \texttt{fl\_polygon() [1/2]}

```c
void fl_polygon (  
    int x,  
    int y,  
    int x1,  
    int y1,  
    int x2,  
    int y2 ) [inline]
```

Fill a 3-sided polygon.

The polygon must be convex.

### 11.7.3.54 \texttt{fl\_polygon() [2/2]}

```c
void fl_polygon (  
    int x,  
    int y,  
    int x1,  
    int y1,  
    int x2,  
    int y2,
```
11.7 Drawing functions

```c
int x3,
int y3) [inline]
```

Fill a 4-sided polygon. The polygon must be convex.

11.7.3.55 fl_pop_clip()

```c
void fl_pop_clip ( ) [inline]
```

Restore the previous clip region. You must call `fl_pop_clip()` once for every time you call `fl_push_clip()`. Unpredictable results may occur if the clip stack is not empty when you return to FLTK.

11.7.3.56 fl_push_clip()

```c
void fl_push_clip ( 
    int x,
    int y,
    int w,
    int h ) [inline]
```

Intersect the current clip region with a rectangle and push this new region onto the stack.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x,y,w,h</th>
<th>position and size</th>
</tr>
</thead>
</table>

11.7.3.57 fl_push_matrix()

```c
void fl_push_matrix ( ) [inline]
```

Save the current transformation matrix on the stack. The maximum depth of the stack is 32.

11.7.3.58 fl_read_image()

```c
uchar * fl_read_image ( 
    uchar * p,
    int X,
    int Y,
    int w,
    int h,
    int alpha )
```

Reads an RGB(A) image from the current window or off-screen buffer.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>p</th>
<th>pixel buffer, or NULL to allocate one</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y</td>
<td>position of top-left of image to read</td>
</tr>
<tr>
<td>in</td>
<td>w,h</td>
<td>width and height of image to read</td>
</tr>
<tr>
<td>in</td>
<td>alpha</td>
<td>alpha value for image (0 for none)</td>
</tr>
</tbody>
</table>

Returns

pointer to pixel buffer, or NULL if allocation failed.

The `p` argument points to a buffer that can hold the image and must be at least `w*h*3` bytes when reading RGB images, or `w*h*4` bytes when reading RGBA images. If NULL, `fl_read_image()` will create an array of the proper size which can be freed using `delete[]`. The `alpha` parameter controls whether an alpha channel is created and the value that is placed in the alpha channel. If 0, no alpha channel is generated.
See also

\texttt{fl\_capture\_window()}

11.7.3.59 \texttt{fl\_rect() [1/3]}

void fl\_rect (\texttt{Fl\_Rect r}) [inline]

Draw a border \textit{inside} the given bounding box.
This is the same as \texttt{fl\_rect(int x, int y, int w, int h)} but with \texttt{Fl\_Rect r} as input argument.

11.7.3.60 \texttt{fl\_rect() [2/3]}

void fl\_rect (int x, int y, int w, int h) [inline]

Draw a border \textit{inside} the given bounding box.
This function is meant for quick drawing of simple boxes. The behavior is undefined for line widths that are not 1.

11.7.3.61 \texttt{fl\_rect() [3/3]}

void fl\_rect (int x, int y, int w, int h, \texttt{Fl\_Color c}) [inline]

Draw with passed color a border \textit{inside} the given bounding box.

Warning

The current color is changed to \texttt{c} upon return.

11.7.3.62 \texttt{fl\_rectf() [1/4]}

void fl\_rectf (\texttt{Fl\_Rect bb, uchar r, uchar g, uchar b}) [inline]

Color a rectangle with “exactly” the passed \texttt{r, g, b} color.
This is the same as \texttt{fl\_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b)} but with \texttt{Fl\_Rect bb} (bounding box) as argument instead of \texttt{(x, y, w, h)}.

See also

\texttt{fl\_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b)}

11.7.3.63 \texttt{fl\_rectf() [2/4]}

void fl\_rectf (\texttt{Fl\_Rect r, Fl\_Color c}) [inline]

Color with passed color a rectangle that exactly fills the given bounding box.

Warning

The current color is changed to \texttt{c} upon return.
11.7.3.64 fl_rectf() [3/4]

```c
void fl_rectf (  
   int x,
   int y,
   int w,
   int h,
   Fl_Color c ) [inline]
```

Color with passed color a rectangle that exactly fills the given bounding box.

Warning

The current color is changed to c upon return.

11.7.3.65 fl_rectf() [4/4]

```c
void fl_rectf (  
   int x,
   int y,
   int w,
   int h,
   uchar r,
   uchar g,
   uchar b ) [inline]
```

Color a rectangle with “exactly” the passed r,g,b color.

On screens with less than 24 bits of color this is done by drawing a solid-colored block using fl_draw_image() so that the correct color shade is produced. On other screens, the current color is changed to fl_color(r,g,b) upon return.

11.7.3.66 fl_rescale_offscreen()

```c
void fl_rescale_offscreen (  
   Fl_Offscreen & ctx)
```

Adapts an offscreen buffer to a changed value of the scale factor.

The ctx argument must have been created by fl_create_offscreen() and the calling context must not be between fl_begin_offscreen() and fl_end_offscreen(). The graphical content of the offscreen is preserved. The current scale factor value is given by Fl_Graphics_Driver::default_driver().scale().

Version

1.4

11.7.3.67 fl_reset_spot()

```c
void fl_reset_spot (  
   void )
```

Resets marked text.

In many languages, typing a character can involve multiple keystrokes. For example, the Ä can be composed of two dots (¨) on top of the character, followed by the letter A (on a Mac with U.S. keyboard, you’d type Alt-U, Shift-A). To inform the user that the dots may be followed by another character, the ¨ is underlined). Call this function if character composition needs to be aborted for some reason. One such example would be the text input widget losing focus.

11.7.3.68 fl_restore_scale()

```c
void fl_restore_scale (  
   float s )
```

Restores the GUI scaling factor and the clipping region in subsequent drawing operations.
### 11.7.3.69 fl_rotate()

```c
void fl_rotate (double d) [inline]
```

Concatenate rotation transformation onto the current one.

**Parameters**

- **d** - rotation angle, counter-clockwise in degrees (not radians)

---

### 11.7.3.70 fl_rounded_rect()

```c
void fl_rounded_rect (int x, int y, int w, int h, int r) [inline]
```

Draw a rounded border inside the given bounding box.

The radius code is optimized for speed and works best for values between 5 and 15 units.

---

### 11.7.3.71 fl_rounded_rectf()

```c
void fl_rounded_rectf (int x, int y, int w, int h, int r) [inline]
```

Color with current color a rounded rectangle that exactly fills the given bounding box.

The radius code is optimized for speed and works best for values between 5 and 15 units.

---

### 11.7.3.72 fl_scale() [1/2]

```c
void fl_scale (double x) [inline]
```

Concatenate scaling transformation onto the current one.

**Parameters**

- **x** - scale factor in both x-direction and y-direction

---

### 11.7.3.73 fl_scale() [2/2]

```c
void fl_scale (double x, double y) [inline]
```

Concatenate scaling transformation onto the current one.

**Parameters**

- **x, y** - scale factors in x-direction and y-direction
11.7 Drawing functions

11.7.3.74 fl_scroll()

```c
void fl_scroll (  
    int X,  
    int Y,  
    int W,  
    int H,  
    int dx,  
    int dy,  
    void(*)(void *, int, int, int, int) draw_area,  
    void * data )
```

Scroll a rectangle and draw the newly exposed portions.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X,Y</td>
<td>position of top-left of rectangle</td>
</tr>
<tr>
<td>W,H</td>
<td>size of rectangle</td>
</tr>
<tr>
<td>dx,dy</td>
<td>pixel offsets for shifting rectangle</td>
</tr>
<tr>
<td>draw_area</td>
<td>callback function to draw rectangular areas</td>
</tr>
<tr>
<td>data</td>
<td>pointer to user data for callback The contents of the rectangular area is first shifted by dx and dy pixels. The draw_area callback is then called for every newly exposed rectangular area.</td>
</tr>
</tbody>
</table>

11.7.3.75 fl_set_spot()

```c
void fl_set_spot (  
    int font,  
    int size,  
    int Y,  
    int W,  
    int H,  
    Fl_Window * win = 0 )
```

Inform text input methods about the current text insertion cursor.

**Parameters**

<table>
<thead>
<tr>
<th>font</th>
<th>Size of the current font.</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>Position of the bottom of the current text insertion cursor.</td>
</tr>
<tr>
<td>X,Y</td>
<td>Width and height of the current text insertion cursor.</td>
</tr>
<tr>
<td>win</td>
<td>Points to the Fl_Window object containing the current text widget, or NULL.</td>
</tr>
</tbody>
</table>

11.7.3.76 fl_set_status()

```c
void fl_set_status (  
    int X,  
    int Y,  
    int W,  
    int H )
```

Related to text input methods under X11.

This function is presently used only by the utf8 test application and only for the X11 platform. This function is apparently not indispensable for text input to work correctly as suggested by other apps that don't use it (e.g., editor).
11.7.3.77  fl_shortcut_label() [1/2]

const char * fl_shortcut_label (  
    unsigned int shortcut  )  

Get a human-readable string from a shortcut value.
Unparse a shortcut value as used by Fl_Button or Fl_Menu_Item into a human-readable string like "Alt+N". This  
only works if the shortcut is a character key or a numbered function key. If the shortcut is zero then an empty string  
is returned. The return value points at a static buffer that is overwritten with each call.  
Since  
FLTK 1.3.4 modifier key names can be localized, but key names can not yet be localized. This may be added  
to a future FLTK version.  
Modifier key names (human-readable shortcut names) can be defined with the following global const char * pointer  
variables:  
• fl_local_ctrl => name of FL_CTRL  
• fl_local_alt => name of FL_ALT  
• fl_local_shift => name of FL_SHIFT  
• fl_local_meta => name of FL_META  
fl_local_ctrl = "Strg";  // German for "Ctrl"  
fl_local_shift = "Umschalt";  // German for "Shift"  

Note  
Due to random static initialization order this should always be done from code in main() or called by main()  
as opposed to static initialization since the default strings in the FLTK library are set by static initializers.  
Otherwise this might result in the wrong order so FLTK's internal initialization overwrites your strings.  
The shortcut name will be constructed by adding all modifier names in the order defined above plus the name of the  
key. A '+' character is added to each modifier name unless it has a trailing '\' or a trailing '+'.  
Example:  
Ctrl+Alt+Shift+Meta+F12  
The default values for modifier key names are as given above for all platforms except macOS. macOS uses graphical  
characters that represent the typical macOS modifier names in menus, e.g. cloverleaf, saucepan, etc. You may,  
however, redefine macOS modifier names as well.  
Parameters  

| in   | shortcut | the integer value containing the ASCII character or extended keystroke plus modifiers |

Returns  
a pointer to a static buffer containing human readable text for the shortcut  

11.7.3.78  fl_shortcut_label() [2/2]

const char * fl_shortcut_label (  
    unsigned int shortcut,  
    const char ** eom  )  

Get a human-readable string from a shortcut value.  
Parameters  

| in   | shortcut | the integer value containing the ASCII character or extended keystroke plus modifiers |
| in   | eom      | if this pointer is set, it will receive a pointer to the end of the modifier text |
Returns

a pointer to a static buffer containing human readable text for the shortcut

See also

fl_shortcut_label(unsigned int shortcut)

11.7.3.79  fl_transform_dx()

double fl_transform_dx (  
    double x,  
    double y)  [inline]

Transform distance using current transformation matrix.

Parameters

| in  | x, y | coordinate |

11.7.3.80  fl_transform_dy()

double fl_transform_dy (  
    double x,  
    double y)  [inline]

Transform distance using current transformation matrix.

Parameters

| in  | x, y | coordinate |

11.7.3.81  fl_transform_x()

double fl_transform_x (  
    double x,  
    double y)  [inline]

Transform coordinate using the current transformation matrix.

Parameters

| in  | x, y | coordinate |

11.7.3.82  fl_transform_y()

double fl_transform_y (  
    double x,  
    double y)  [inline]

Transform coordinate using the current transformation matrix.

Parameters

| in  | x, y | coordinate |
11.7.3.83 fl_transformed_vertex()

```c
void fl_transformed_vertex (  
    double xf,  
    double yf ) [inline]
```

Add coordinate pair to the vertex list without further transformations.

**Parameters**

- `in` `xf, yf` transformed coordinate

11.7.3.84 fl_translate()

```c
void fl_translate (  
    double x,  
    double y  ) [inline]
```

Concatenate translation transformation onto the current one.

**Parameters**

- `in` `x, y` translation factor in x-direction and y-direction

11.7.3.85 fl_vertex()

```c
void fl_vertex (  
    double x,  
    double y  ) [inline]
```

Add a single vertex to the current path.

**Parameters**

- `in` `x, y` coordinate

11.8 Multithreading support functions

fl multithreading support functions declared in `<FL/Fl.H>`

**Functions**

- `static int Fl::awake (Fl_Awake_Handler cb, void *message=0)`  
  See void awake(void *message=0).
- `static void Fl::awake (void *message=0)`  
  Sends a message pointer to the main thread, causing any pending Fl::wait() call to terminate so that the main thread can retrieve the message and any pending redraws can be processed.
- `static int Fl::lock ()`  
  The lock() method blocks the current thread until it can safely access FLTK widgets and data.
- `static void * Fl::thread_message ()`  
  The thread_message() method returns the last message that was sent from a child by the awake() method.
- `static void Fl::unlock ()`  
  The unlock() method releases the lock that was set using the lock() method.

11.8.1 Detailed Description

fl multithreading support functions declared in `<FL/Fl.H>`
11.8.2 Function Documentation

11.8.2.1 awake() [1/2]

```c
int Fl::awake (  
    Fl_Awake_Handler func,  
    void * data = 0 ) [static]
```

See void awake(void * message=0).

Let the main thread know an update is pending and have it call a specific function.

Registers a function that will be called by the main thread during the next message handling cycle. Returns 0 if the callback function was registered, and -1 if registration failed. Over a thousand awake callbacks can be registered simultaneously.

See also

- Fl::awake(void * message=0)

11.8.2.2 awake() [2/2]

```c
void Fl::awake (  
    void * msg = 0 ) [static]
```

Sends a message pointer to the main thread, causing any pending Fl::wait() call to terminate so that the main thread can retrieve the message and any pending redraws can be processed.

Multiple calls to Fl::awake() will queue multiple pointers for the main thread to process, up to a system-defined (typically several thousand) depth. The default message handler saves the last message which can be accessed using the Fl::thread_message() function.

In the context of a threaded application, a call to Fl::awake() with no argument will trigger event loop handling in the main thread. Since it is not possible to call Fl::flush() from a subsidiary thread, Fl::awake() is the best (and only, really) substitute.

It's not necessary to wrap calls to any form of Fl::awake() by Fl::lock() and Fl::unlock(). Nevertheless, the early, single call to Fl::lock() used to initialize threading support is necessary.

Function Fl::awake() in all its forms is typically called by worker threads, but it can be used safely by the main thread too, as a means to break the event loop.

See also

- Multithreading

11.8.2.3 lock()

```c
int Fl::lock ( ) [static]
```

The lock() method blocks the current thread until it can safely access FLTK widgets and data.

Child threads should call this method prior to updating any widgets or accessing data. The main thread must call lock() to initialize the threading support in FLTK. lock() will return non-zero if threading is not available on the platform.

Child threads must call unlock() when they are done accessing FLTK.

When the wait() method is waiting for input or timeouts, child threads are given access to FLTK. Similarly, when the main thread needs to do processing, it will wait until all child threads have called unlock() before processing additional data.

Returns

- 0 if threading is available on the platform; non-zero otherwise

See also: Multithreading

11.8.2.4 thread_message()

```c
void * Fl::thread_message ( ) [static]
```

The thread_message() method returns the last message that was sent from a child by the awake() method.

See also: Multithreading
11.8.2.5 unlock()

```cpp
void Fl::unlock ( ) [static]
```

The `unlock()` method releases the lock that was set using the `lock()` method. Child threads should call this method as soon as they are finished accessing FLTK.

See also: Multithreading

11.9 Safe widget deletion support functions

These functions, declared in `<FL/Fl.H>`, support deletion of widgets inside callbacks.

**Functions**

- static void `Fl::clear_widget_pointer (Fl_Widget const *w)`
  
  Clears a widget pointer in the watch list.

- static void `Fl::delete_widget (Fl_Widget *w)`
  
  Schedules a widget for deletion at the next call to the event loop.

- static void `Fl::do_widget_deletion ()`
  
  Deletes widgets previously scheduled for deletion.

- static void `Fl::release_widget_pointer (Fl_Widget *&w)`
  
  Removes a widget pointer from the watch list.

- static void `Fl::watch_widget_pointer (Fl_Widget *&w)`
  
  Adds a widget pointer to the widget watch list.

11.9.1 Detailed Description

These functions, declared in `<FL/Fl.H>`, support deletion of widgets inside callbacks. `Fl::delete_widget()` should be called when deleting widgets or complete widget trees (`Fl_Group`, `Fl_Window`, ...) inside callbacks.

The other functions are intended for internal use. The preferred way to use them is by using the helper class `Fl_Widget_Tracker`.

The following is to show how it works ...

There are three groups of related methods:

1. scheduled widget deletion

   - `Fl::delete_widget()` schedules widgets for deletion
   - `Fl::do_widget_deletion()` deletes all scheduled widgets

2. widget watch list ("smart pointers")

   - `Fl::watch_widget_pointer()` adds a widget pointer to the watch list
   - `Fl::release_widget_pointer()` removes a widget pointer from the watch list
   - `Fl::clear_widget_pointer()` clears a widget pointer *in* the watch list

3. the class `Fl_Widget_Tracker`:

   - the constructor calls `Fl::watch_widget_pointer()`
   - the destructor calls `Fl::release_widget_pointer()`
   - the access methods can be used to test, if a widget has been deleted

See also: `Fl_Widget_Tracker`. 
11.9 Safe widget deletion support functions

11.9.2 Function Documentation

11.9.2.1 clear_widget_pointer()

```c
void Fl::clear_widget_pointer ( Fl_Widget const * w ) [static]
```

Clears a widget pointer in the watch list. This is called when a widget is destroyed (by its destructor). You should never call this directly.

**Note**

Internal use only!

This method searches the widget watch list for pointers to the widget and clears each pointer that points to it. Widget pointers can be added to the widget watch list by calling Fl::watch_widget_pointer() or by using the helper class Fl_Widget_Tracker (recommended).

See also

Fl::watch_widget_pointer()
class Fl_Widget_Tracker

11.9.2.2 delete_widget()

```c
void Fl::delete_widget ( Fl_Widget * wi ) [static]
```

Schedules a widget for deletion at the next call to the event loop. Use this method to delete a widget inside a callback function.

To avoid early deletion of widgets, this function should be called toward the end of a callback and only after any call to the event loop (Fl::wait(), Fl::flush(), Fl::check(), fl_ask(), etc.). When deleting groups or windows, you must only delete the group or window widget and not the individual child widgets.

Since

FLTK 1.3.4 the widget will be hidden immediately, but the actual destruction will be delayed until the event loop is finished. Up to FLTK 1.3.3 windows wouldn't be hidden before the event loop was done, hence you had to hide() a window in your window close callback if you called Fl::delete_widget() to destroy (and hide) the window.

FLTK 1.3.0 it is not necessary to remove widgets from their parent groups or windows before calling this, because it will be done in the widget's destructor, but it is not a failure to do this nevertheless.

**Note**

In FLTK 1.1 you must remove widgets from their parent group (or window) before deleting them.

See also

Fl_Widget::~Fl_Widget()

11.9.2.3 do_widget_deletion()

```c
void Fl::do_widget_deletion ( ) [static]
```

Deletes widgets previously scheduled for deletion. This is for internal use only. You should never call this directly.

Fl::do_widget_deletion() is called from the FLTK event loop or whenever you call Fl::wait(). The previously scheduled widgets are deleted in the same order they were scheduled by calling Fl::delete_widget().

See also

Fl::delete_widget(Fl_Widget +wi)
11.9.2.4 release_widget_pointer()

```cpp
void Fl::release_widget_pointer ( 
    Fl_Widget *& w ) [static]
```
Releases a widget pointer from the watch list.
This is used to remove a widget pointer that has been added to the watch list with Fl::watch_widget_pointer(), when it is not needed anymore.

**Note**

Internal use only, please use class Fl_Widget_Tracker instead.

See also

Fl::watch_widget_pointer()

11.9.2.5 watch_widget_pointer()

```cpp
void Fl::watch_widget_pointer ( 
    Fl_Widget *& w ) [static]
```
Adds a widget pointer to the widget watch list.

**Note**

Internal use only, please use class Fl_Widget_Tracker instead.

This can be used, if it is possible that a widget might be deleted during a callback or similar function. The widget pointer must be added to the watch list before calling the callback. After the callback the widget pointer can be queried, if it is NULL. If it is NULL, then the widget has been deleted during the callback and must not be accessed anymore. If the widget pointer is not NULL, then the widget has not been deleted and can be accessed safely. After accessing the widget, the widget pointer must be released from the watch list by calling Fl::release_widget_pointer().

Example for a button that is clicked (from its handle() method):

```cpp
Fl_Widget *wp = this; // save 'this' in a pointer variable
Fl::watch_widget_pointer(wp); // add the pointer to the watch list
set_changed(); // set the changed flag
do_callback(); // call the callback
if (!wp) { // the widget has been deleted
    // DO NOT ACCESS THE DELETED WIDGET !
} else { // the widget still exists
    clear_changed(); // reset the changed flag
}
Fl::release_widget_pointer(wp); // remove the pointer from the watch list
```
This works, because all widgets call Fl::clear_widget_pointer() in their destructors.

See also

Fl::release_widget_pointer()  
Fl::clear_widget_pointer()

An easier and more convenient method to control widget deletion during callbacks is to use the class Fl_Widget_Tracker with a local (automatic) variable.

See also

class Fl_Widget_Tracker

11.10 Cairo Support Functions and Classes

**Classes**

- class Fl_Cairo_State
  
  Contains all the necessary info on the current cairo context.

- class Fl_Cairo_Window
  
  This defines an FLTK window with Cairo support.
Functions

- static bool Fl::cairo_autolink_context ()
  Gets the current autolink mode for Cairo support.

- static void Fl::cairo_autolink_context (bool alink)
  When FLTK_HAVE_CAIRO is defined and cairo_autolink_context() is true, any current window dc is linked to a current Cairo context.

- static cairo_t * Fl::cairo_cc ()
  Gets the current Cairo context linked with a fltk window.

- static void Fl::cairo_cc (cairo_t *c, bool own=false)
  Sets the current Cairo context to c.

- static void Fl::cairo_flush (cairo_t *c)
  Flush Cairo drawings on Cairo context c.

- static cairo_t * Fl::cairo_make_current (Fl_Window *w)
  Provides a Cairo context for window w.

11.10.1 Detailed Description

11.10.2 Function Documentation

11.10.2.1 cairo_autolink_context() [1/2]

static bool Fl::cairo_autolink_context () [inline], [static]
Gets the current autolink mode for Cairo support.

Return values

| false | if no Cairo context autolink is made for each window. |
| true  | if any fltk window is attached a Cairo context when it is current. |

See also

void cairo_autolink_context(bool alink)

Note

Only available when configure has the –enable-cairo option

11.10.2.2 cairo_autolink_context() [2/2]

static void Fl::cairo_autolink_context (bool alink) [inline], [static]
When FLTK_HAVE_CAIRO is defined and cairo_autolink_context() is true, any current window dc is linked to a current Cairo context.

This is not the default, because it may not be necessary to add Cairo support to all fltk supported windows. When you wish to associate a Cairo context in this mode, you need to call explicitly in your draw() overridden method, Fl::cairo_make_current(Fl_Window *w). This will create a Cairo context only for this Window. Still in custom Cairo application it is possible to handle completely this process automatically by setting alink to true. In this last case, you don't need anymore to call Fl::cairo_make_current(). You can use Fl::cairo_cc() to get the current Cairo context anytime.

Note

Only available when configure has the –enable-cairo option
### 11.10.2.3 cairo_cc()

```cpp
class Fl
{
public:
  static void cairo_cc (cairo_t * c, bool own = false) [inline], [static]
}
```

Sets the current Cairo context to `c`. Set `own` to true if you want FLTK to handle this cc deletion.

**Note**

Only available when configure has the --enable-Cairo option.

### 11.10.2.4 cairo_flush()

```cpp
class Fl
{
public:
  static void cairo_flush (cairo_t * c) [inline], [static]
}
```

Flush Cairo drawings on Cairo context `c`. This is **required** on Windows if you use the Cairo context provided by the "Cairo autolink" option. Call this when all your drawings on the Cairo context are finished. This is maybe not necessary on other platforms than Windows but it does no harm if you call it always.

You don't need to use this if you use an `Fl_Cairo_Window` which does this automatically after the draw callback returns.

**Code example for "Cairo autolink" mode:**

In the overridden draw() method of your subclass of `Fl_Window` or any widget:

```cpp
cairo_t *cc = Fl::cairo_cc(); // get the "autolink" Cairo context
// ... your Cairo drawings are here ...
Fl::cairo_flush(cc); // flush Cairo drawings to the device
```

If you configure FLTK with '--enable-cairo' or CMake option 'OPTION_CAIRO' (i.e. without '--enable-cairoext' or CMake option 'OPTION_CAIROEXT') or if you don't enable the 'autolink' Cairo context you may do the equivalent to use Cairo drawings in an overridden draw() method of derived classes by using

```cpp
// get the Cairo context for the \c window
cairo_t *cc = Fl::cairo_make_current(window);
// ... your Cairo drawings are here ...
Fl::cairo_flush(cc); // flush Cairo drawings to the device
```

See also

- `Fl::cairo_autolink_context(bool)`
- `Fl::cairo_make_current(Fl_Window*)`;

### 11.10.2.5 cairo_make_current()

```cpp
cairo_t *Fl::cairo_make_current (Fl_Window * wi) [static]
```

Provides a Cairo context for window `wi`. This is needed in a draw() override if `Fl::cairo_autolink_context()` returns false, which is the default. The `cairo_make_current()` does not need to be freed as it is freed every time a new Cairo context is created. When the program terminates, a call to `Fl::cairo_make_current(0)` will destroy any residual context.

**Note**

A new Cairo context is not always re-created when this method is used. In particular, if the current graphical context and the current window didn't change between two calls, the previous gc is internally kept, thus optimizing the drawing performances. Also, after this call, `Fl::cairo_cc()` is adequately updated with this Cairo context.

Only available when configure has the --enable-cairo option

**Returns**

The valid cairo_t *cairo context associated to this window.
11.11 Unicode and UTF-8 functions

fl global Unicode and UTF-8 handling functions declared in `<FL/fl_utf8.h>`

**Macros**

- `#define ERRORS_TO_CP1252 1`
  
  Set to 1 to turn bad UTF-8 bytes in the 0x80-0x9f range into the Unicode index for Microsoft’s CP1252 character set.

- `#define ERRORS_TO_ISO8859_1 1`
  
  Set to 1 to turn bad UTF-8 bytes into ISO-8859-1.

- `#define NBC 0xFFFF + 1`

- `#define STRICT_RFC3629 0`
  
  A number of Unicode code points are in fact illegal and should not be produced by a UTF-8 converter.

**Functions**

- `int fl_access (const char *f, int mode)`
  
  Cross-platform function to test a files access() with a UTF-8 encoded name or value.

- `int fl_chdir (const char *path)`
  
  Cross-platform function to change the current working directory, given as a UTF-8 encoded string.

- `int fl_chmod (const char *f, int mode)`
  
  Cross-platform function to set a files mode() with a UTF-8 encoded name or value.

- `int fl_close_fd (int fd)`

- `int fl_execvp (const char *file, char **argv)`

- `FILE *fl_fopen (const char *f, const char *mode)`
  
  Cross-platform function to open files with a UTF-8 encoded name.

- `char *fl_getcwd (char *buf, int len)`
  
  Cross-platform function to get the current working directory as a UTF-8 encoded value.

- `char *fl_getenv (const char *v)`
  
  Cross-platform function to get environment variables with a UTF-8 encoded name or value.

- `char *fl_make_path (const char *path)`
  
  Cross-platform function to recursively create a path in the file system.

- `void fl_make_path_for_file (const char *path)`
  
  Cross-platform function to create a path for the file in the file system.

- `int fl_mtime (const char *f, int mode)`
  
  Cross-platform function to set a files modification time with a UTF-8 encoded name or value.

- `unsigned int fl_nonspacing (unsigned int ucs)`
  
  Returns true if the Unicode character `ucs` is non-spacing.

- `int fl_open (const char *fname, int oflags,...)`
  
  Cross-platform function to open files with a UTF-8 encoded name.

- `int fl_putenv (const char *var)`
  
  Cross-platform function to write environment variables with a UTF-8 encoded name or value.

- `int fl_rename (const char *f, const char *n)`
  
  Cross-platform function to rename a filesystem object using UTF-8 encoded names.
• int fl_rmdir (const char *f)
  Cross-platform function to remove a directory with a UTF-8 encoded name.

• int fl_stat (const char *f, struct stat *s)
  Cross-platform function to stat() a file using a UTF-8 encoded name or value.

• int fl_system (const char *cmd)
  Cross-platform function to run a system command with a UTF-8 encoded string.

• int fl_tolower (unsigned int ucs)
  Returns the Unicode lower case value of `ucs`.

• int fl_toupper (unsigned int ucs)
  Returns the Unicode upper case value of `ucs`.

• unsigned fl_ucs_to_Utf16 (const unsigned ucs, unsigned short *dst, const unsigned dstlen)
  Convert a single 32-bit Unicode codepoint into an array of 16-bit characters.

• int fl_unlink (const char *fname)
  Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename.

• char * fl_utf2mbcs (const char *s)
  Converts UTF-8 string `s` to a local multi-byte character string.

• const char * fl_utf8back (const char *p, const char *start, const char *end)
  Move `p` backward until it points to the start of a UTF-8 character.

• int fl_utf8bytes (unsigned ucs)
  Return the number of bytes needed to encode the given UCS4 character in UTF-8.

• unsigned fl_utf8decode (const char *p, const char *end, int *len)
  Decode a single UTF-8 encoded character starting at `p`.

• int fl_utf8encode (unsigned ucs, char *buf)
  Write the UTF-8 encoding of `ucs` into `buf` and return the number of bytes written.

• unsigned fl_utf8from_mb (char *dst, unsigned dstlen, const char *src, unsigned srclen)
  Convert a filename from the locale-specific multibyte encoding used by Windows to UTF-8 as used by FLTK.

• unsigned fl_utf8froma (const char *src, unsigned srclen, char *dst, unsigned dstlen)
  Convert an ISO-8859-1 (ie normal c-string) byte stream to UTF-8.

• unsigned fl_utf8fromwc (char *dst, unsigned dstlen, const wchar_t *src, unsigned srclen)
  Turn "wide characters" as returned by some system calls (especially on Windows) into UTF-8.

• const char * fl_utf8fwd (const char *p, const char *start, const char *end)
  Move `p` forward until it points to the start of a UTF-8 character.

• int fl_utf8fslen (char c)
  Returns the byte length of the UTF-8 sequence with first byte `c`, or -1 if `c` is not valid.

• int fl_utf8len1 (char c)
  Returns the byte length of the UTF-8 sequence with first byte `c`, or 1 if `c` is not valid.

• int fl_utf8locale ()
  Return true if the "locale" seems to indicate that UTF-8 encoding is used.

• int fl_utf8strlen (const char *text, int len)
  Return the length in bytes of a UTF-8 string.

• int fl_utf8test (const char *src, unsigned srclen)
  Examines the first `srclen` bytes in `src` and returns a verdict on whether it is UTF-8 or not.

• unsigned fl_utf8to_mb (const char *src, unsigned srclen, char *dst, unsigned dstlen)
  Convert the UTF-8 used by FLTK to the locale-specific encoding used for filenames (and sometimes used for data in files).

• unsigned fl_utf8toa (const char *src, unsigned srclen, char *dst, unsigned dstlen)
  Convert a UTF-8 sequence into an array of 1-byte characters.

• unsigned fl_utf8toUtf16 (const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen)
  Convert a UTF-8 sequence into an array of 16-bit characters.

• unsigned fl_utf8lowc (const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen)
  Converts a UTF-8 string into a wide character string.
11.11 Unicode and UTF-8 functions

- `int fl_utfNbChar (const unsigned char *buf, int len)`
  Returns the number of Unicode chars in the UTF-8 string.

- `int fl_utfStrcasecmp (const char *s1, const char *s2)`
  UTF-8 aware strcasecmp - converts to Unicode and tests.

- `int fl_utfStrncasecmp (const char *s1, const char *s2, int n)`
  UTF-8 aware strncasecmp - converts to lower case Unicode and tests.

- `int fl_utfToLower (const unsigned char *str, int len, char *buf)`
  Converts the string `str` to its lower case equivalent into `buf`.

- `int fl_utfToUpper (const unsigned char *str, int len, char *buf)`
  Converts the string `str` to its upper case equivalent into `buf`.

- `int fl_wcwidth (const char *src)`
  extended wrapper around `fl_wcwidth_(unsigned int ucs)` function.

- `int fl_wcwidth_(unsigned int ucs)`
  Wrapper to adapt Markus Kuhn's implementation of wcwidth() for FLTK.

11.11.1 Detailed Description

fl global Unicode and UTF-8 handling functions declared in `<FL/fl_utf8.h>`

11.11.2 Macro Definition Documentation

11.11.2.1 ERRORS_TO_CP1252

#define ERRORS_TO_CP1252 1
Set to 1 to turn bad UTF-8 bytes in the 0x80-0x9f range into the Unicode index for Microsoft's CP1252 character set.
You should also set ERRORS_TO_ISO8859_1. With this a huge amount of more available text (such as all web pages) are correctly converted to Unicode.

11.11.2.2 ERRORS_TO_ISO8859_1

#define ERRORS_TO_ISO8859_1 1
Set to 1 to turn bad UTF-8 bytes into ISO-8859-1.
If this is zero they are instead turned into the Unicode REPLACEMENT CHARACTER, of value 0xfffd. If this is on `fl_utf8decode()` will correctly map most (perhaps all) human-readable text that is in ISO-8859-1. This may allow you to completely ignore character sets in your code because virtually everything is either ISO-8859-1 or UTF-8.

11.11.2.3 STRICT_RFC3629

#define STRICT_RFC3629 0
A number of Unicode code points are in fact illegal and should not be produced by a UTF-8 converter.
Turn this on will replace the bytes in those encodings with errors. If you do this then converting arbitrary 16-bit data to UTF-8 and then back is not an identity, which will probably break a lot of software.

11.11.3 Function Documentation

11.11.3.1 fl_access()

    int fl_access (const char * f, int mode )

Cross-platform function to test a files access() with a UTF-8 encoded name or value.
This function is especially useful on the Windows platform where the standard access() function fails with UTF-8 encoded non-ASCII filenames.
Windows defines the mode values 0 for existence, 2 for writable, 4 for readable, and 6 of readable and writable. On other systems, the modes X_OK, W_OK, and R_OK are usually defined as 1, 2, and 4.
Upon successful completion, the value 0 is returned on all platforms.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>f</th>
<th>the UTF-8 encoded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mode</td>
<td>the mode to test</td>
</tr>
</tbody>
</table>

Returns

the return value of _waccess() on Windows or access() on other platforms.

11.11.3.2 fl_chdir()

```c
int fl_chdir (const char * path )
```

Cross-platform function to change the current working directory, given as a UTF-8 encoded string. This function is especially useful on the Windows platform where the standard _wchdir() function needs a path in UTF-16 encoding. The path is converted to a system specific encoding if necessary and the system specific chdir(converted←_path) function is called. The function returns 0 on success and -1 on error. Depending on the platform, errno may be set if an error occurs.

Note

The possible errno values are platform specific. Refer to the documentation of the platform specific chdir() function.

If the function is not implemented on a particular platform the default implementation returns -1 and errno is not set.

If the path is NULL the function returns -1, but errno is not changed. This is a convenience feature of fl_chdir() as opposed to chdir().

Parameters

| in  | path    | the target directory for chdir (may be NULL) |

Returns

0 if successful, -1 on error (errno may be set)

11.11.3.3 fl_chmod()

```c
int fl_chmod (const char * f, int mode )
```

Cross-platform function to set a files mode() with a UTF-8 encoded name or value. This function is especially useful on the Windows platform where the standard chmod() function fails with UTF-8 encoded non-ASCII filenames.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>f</th>
<th>the UTF-8 encoded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mode</td>
<td>the mode to set</td>
</tr>
</tbody>
</table>
11.11 Unicode and UTF-8 functions

Returns

the return value of _wchmod() on Windows or chmod() on other platforms.

11.11.3.4  fl_close_fd()

```c
int fl_close_fd (  
    int fd  
)
```

Cross-platform function to close a file descriptor.

Returns

0 in case of success, or -1 in case of error.

11.11.3.5  fl_fopen()

```c
FILE * fl_fopen (  
    const char * f,  
    const char * mode  
)
```

Cross-platform function to open files with a UTF-8 encoded name.
This function is especially useful on the Windows platform where the standard fopen() function fails with UTF-8 encoded non-ASCII filenames.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>f</strong></td>
<td>the UTF-8 encoded filename</td>
</tr>
<tr>
<td><strong>mode</strong></td>
<td>same as the second argument of the standard fopen() function</td>
</tr>
</tbody>
</table>

Returns

a FILE pointer upon successful completion, or NULL in case of error.

See also

fl_open().

11.11.3.6  fl_getcwd()

```c
char * fl_getcwd (  
    char * buf,  
    int len  
)
```

Cross-platform function to get the current working directory as a UTF-8 encoded value.
This function is especially useful on the Windows platform where the standard _wgetcwd() function returns UTF-16 encoded non-ASCII filenames.
If **buf** is NULL a buffer of size (len+1) is allocated, filled with the current working directory, and returned. In this case the buffer must be released by the caller with free() to prevent memory leaks.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>buf</strong></td>
<td>the buffer to populate (may be NULL)</td>
</tr>
<tr>
<td><strong>len</strong></td>
<td>the length of the buffer</td>
</tr>
</tbody>
</table>
Returns

the CWD encoded as UTF-8

### 11.11.3.7 fl getenv()

```c
char * fl_getenv ( const char * v )
```

Cross-platform function to get environment variables with a UTF-8 encoded name or value.

This function is especially useful on the Windows platform where non-ASCII environment variables are encoded as wide characters. The returned value of the variable is encoded in UTF-8 as well.

On platforms other than Windows this function calls getenv directly. The return value is returned as-is.

The return value is a pointer to an implementation defined buffer:

- an internal buffer that is (re)allocated as needed (Windows) or
- the string in the environment itself (Unix, Linux, MacOS) or
- any other implementation (other platforms). This string must be considered read-only and must not be freed by the caller.

If the resultant string is to be used later it must be copied to a safe place. The next call to `fl_getenv()` or any other environment changes may overwrite the string.

**Note**

This function is not thread-safe.

**Parameters**

`v` the UTF-8 encoded environment variable

**Returns**

the environment variable in UTF-8 encoding, or NULL in case of error.

### 11.11.3.8 fl make_path()

```c
char fl_make_path ( const char * path )
```

Cross-platform function to recursively create a path in the file system.

This function creates a path in the file system by recursively creating all directories.

**Parameters**

`path` a Unix style (" forward slashes) absolute or relative pathname

**Returns**

1 if the path was created, 0 if creating the path failed at some point

### 11.11.3.9 fl make_path_for_file()

```c
void fl_make_path_for_file ( const char * path )
```

Cross-platform function to create a path for the file in the file system.

This function strips the filename from the given path and creates a path in the file system by recursively creating all directories.
11.11.3.10 \textbf{fl_mkdir()}

\begin{verbatim}
int fl_mkdir (  
    const char * f,  
    int mode  
)
\end{verbatim}

Cross-platform function to create a directory with a UTF-8 encoded name. This function is especially useful on the Windows platform where the standard \_wmkdir() function expects UTF-16 encoded non-ASCII filenames.

\textbf{Parameters}

\begin{tabular}{|c|c|}
\hline
in & \textit{f} & the UTF-8 encoded filename \\
\hline
in & \textit{mode} & the mode of the directory \\
\hline
\end{tabular}

\textbf{Returns}

the return value of \_wmkdir() on Windows or mkdir() on other platforms.

11.11.3.11 \textbf{fl_nonspacing()}

\begin{verbatim}
unsigned int fl_nonspacing (  
    unsigned int ucs  
)
\end{verbatim}

Returns true if the Unicode character \texttt{ucs} is non-spacing. Non-spacing characters in Unicode are typically combining marks like tilde (~), diaeresis (¨), or other marks that are added to a base character, for instance ‘a’ (base character) + ‘¨’ (combining mark) = ‘ä’ (German Umlaut).

\begin{itemize}
  \item http://unicode.org/glossary/#base_character
  \item http://unicode.org/glossary/#nonspacing_mark
  \item http://unicode.org/glossary/#combining_character
\end{itemize}

11.11.3.12 \textbf{fl_open()}

\begin{verbatim}
int fl_open (  
    const char * fname,  
    int oflags,  
    ...  
)
\end{verbatim}

Cross-platform function to open files with a UTF-8 encoded name. This function is especially useful on the Windows platform where the standard \texttt{open()} function fails with UTF-8 encoded non-ASCII filenames.

\textbf{Parameters}

\begin{tabular}{|c|c|}
\hline
in & \textit{fname} & the UTF-8 encoded filename \\
\hline
in & \textit{oflags} & other arguments are as in the standard \texttt{open()} function \\
\hline
\end{tabular}

\textbf{Returns}

a file descriptor upon successful completion, or -1 in case of error.

\textbf{See also}

\texttt{fl_fopen()}, \texttt{fl_open_ext()}, \texttt{fl_close_fd(int fd)}.

11.11.3.13 \textbf{fl_open_ext()}

\begin{verbatim}
int fl_open_ext (  
    const char * fname,  
    ...  
)
\end{verbatim}
Cross-platform function to open files with a UTF-8 encoded name.
In comparison with \texttt{fl_open()}, this function allows to control whether the file is opened in binary (a.k.a. untranslated) mode. This is especially useful on the Windows platform where files are by default opened in text (translated) mode.

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{int}</td>
<td>\texttt{fname}</td>
<td>the UTF-8 encoded filename</td>
</tr>
<tr>
<td>\texttt{int}</td>
<td>\texttt{binary}</td>
<td>if non-zero, the file is to be accessed in binary (a.k.a. untranslated) mode.</td>
</tr>
<tr>
<td>...</td>
<td>flags,...</td>
<td>these arguments are as in the standard open() function. Setting flags to zero opens the file for reading.</td>
</tr>
</tbody>
</table>

### Returns

A file descriptor upon successful completion, or -1 in case of error.

---

11.11.3.14 \texttt{fl_putenv()}

\begin{verbatim}
int fl_putenv (const char * var )
\end{verbatim}

Cross-platform function to write environment variables with a UTF-8 encoded name or value.
This function is especially useful on the Windows platform where non-ASCII environment variables are encoded as wide characters.
The given argument \texttt{var} must be encoded in UTF-8 in the form "name=value". The 'name' part must conform to platform dependent restrictions on environment variable names. The string given in \texttt{var} is copied and optionally converted to the required encoding for the platform. On platforms other than Windows this function calls putenv directly. The return value is zero on success and non-zero in case of error. The value in case of error is platform specific and returned as-is.

Note

The copied string is allocated on the heap and "lost" on some platforms, i.e. calling \texttt{fl_putenv()} to change environment variables frequently may cause memory leaks. There may be an option to avoid this in a future implementation.

This function is not thread-safe.

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{in}</td>
<td>\texttt{var}</td>
<td>the UTF-8 encoded environment variable 'name=value'</td>
</tr>
</tbody>
</table>

### Returns

0 on success, non-zero in case of error.

---

11.11.3.15 \texttt{fl_rename()}

\begin{verbatim}
int fl_rename (const char * f, const char * n )
\end{verbatim}

Cross-platform function to rename a filesystem object using UTF-8 encoded names.
This function is especially useful on the Windows platform where the standard \_wrename() function expects UTF-16 encoded non-ASCII filenames.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>f</th>
<th>the UTF-8 encoded filename to change</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>the new UTF-8 encoded filename to set</td>
</tr>
</tbody>
</table>

Returns

the return value of _wrename() on Windows or rename() on other platforms.

11.11.3.16 fl_rmdir()

```c
int fl_rmdir (const char * f)
```

Cross-platform function to remove a directory with a UTF-8 encoded name.
This function is especially useful on the Windows platform where the standard _wrmdir() function expects UTF-16
encoded non-ASCII filenames.

Parameters

| in  | f   | the UTF-8 encoded filename to remove |

Returns

the return value of _wrmdir() on Windows or rmdir() on other platforms.

11.11.3.17 fl_stat()

```c
int fl_stat (const char * f, struct stat * b)
```

Cross-platform function to stat() a file using a UTF-8 encoded name or value.
This function is especially useful on the Windows platform where the standard stat() function fails with UTF-8 en-
coded non-ASCII filenames.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>f</th>
<th>the UTF-8 encoded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>b</td>
<td>the stat struct to populate</td>
</tr>
</tbody>
</table>

Returns

the return value of _wstat() on Windows or stat() on other platforms.

11.11.3.18 fl_system()

```c
int fl_system (const char * cmd)
```

Cross-platform function to run a system command with a UTF-8 encoded string.
This function is especially useful on the Windows platform where non-ASCII program (file) names must be encoded
as wide characters.
On platforms other than Windows this function calls system() directly.

Parameters

| in  | cmd | the UTF-8 encoded command string |

Generated by Doxygen
Returns

the return value of _wsystem() on Windows or system() on other platforms.

11.11.3.19 fl_ucs_to_Utf16()

unsigned fl_ucs_to_Utf16 (
    const unsigned ucs,
    unsigned short * dst,
    const unsigned dstlen )

Convert a single 32-bit Unicode codepoint into an array of 16-bit characters. These are used by some system calls, especially on Windows.

`ucs` is the value to convert.
`dst` points at an array to write, and `dstlen` is the number of locations in this array. At most `dstlen` words will be written, and a 0 terminating word will be added if `dstlen` is large enough. Thus this function will never overwrite the buffer and will attempt return a zero-terminated string if space permits. If `dstlen` is zero then `dst` can be set to NULL and no data is written, but the length is returned.

The return value is the number of 16-bit words that _would_ be written to `dst` if it is large enough, not counting any terminating zero.

If the return value is greater than `dstlen` it indicates truncation, you should then allocate a new array of size `return+1` and call this again.

Unicode characters in the range 0x10000 to 0x10ffff are converted to "surrogate pairs" which take two words each (in UTF-16 encoding). Typically, setting `dstlen` to 2 will ensure that any valid Unicode value can be converted, and setting `dstlen` to 3 or more will allow a NULL terminated sequence to be returned.

11.11.3.20 fl_unlink()

int fl_unlink (
    const char * fname )

Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename. This function is especially useful on the Windows platform where the standard function expects UTF-16 encoded non-ASCII filenames.

Parameters

| fname | the filename to unlink |

Returns

the return value of _wunlink() on Windows or unlink() on other platforms.

11.11.3.21 fl_utf8back()

const char * fl_utf8back ( 
    const char * p,
    const char * start,
    const char * end )

Move `p` backward until it points to the start of a UTF-8 character.
If it already points at the start of one then it is returned unchanged. Any UTF-8 errors are treated as though each byte of the error is an individual character.
`start` is the start of the string and is used to limit the backwards search for the start of a UTF-8 character.
`end` is the end of the string and is assumed to be a break between characters. It is assumed to be greater than `p`.
If you wish to decrement a UTF-8 pointer, pass `p-1` to this.

11.11.3.22 fl_utf8bytes()

int fl_utf8bytes ( 
    unsigned ucs )

Generated by Doxygen
11.11 Unicode and UTF-8 functions

Return the number of bytes needed to encode the given UCS4 character in UTF-8. Returns number of bytes that utf8encode() will use to encode the character ucs.

Parameters

| in  | ucs | UCS4 encoded character |

Returns

number of bytes required

11.11.3.23 fl_utf8decode()

```c
unsigned fl_utf8decode {
    const char * p,
    const char * end,
    int * len }
```

Decode a single UTF-8 encoded character starting at p. The resulting Unicode value (in the range 0-0x10ffff) is returned, and len is set to the number of bytes in the UTF-8 encoding (adding len to p will point at the next character).

If p points at an illegal UTF-8 encoding, including one that would go past end, or where a code uses more bytes than necessary, then *(unsigned char *)p is translated as though it is in the Microsoft CP1252 character set and len is set to 1. Treating errors this way allows this to decode almost any ISO-8859-1 or CP1252 text that has been mistakenly placed where UTF-8 is expected, and has proven very useful.

If you want errors to be converted to error characters (as the standards recommend), adding a test to see if the length is unexpectedly 1 will work:

```c
if (*p & 0x80) { // what should be a multibyte encoding
    code = fl_utf8decode(p,end,len);
    if (len<2) code = 0xFFFD; // Turn errors into REPLACEMENT CHARACTER
    else { // handle the 1-byte UTF-8 encoding:
        code = *p;
        len = 1;
    }
}
```

Direct testing for the 1-byte case (as shown above) will also speed up the scanning of strings where the majority of characters are ASCII.

11.11.3.24 fl_utf8encode()

```c
int fl_utf8encode {
    unsigned ucs,
    char * buf }
```

Write the UTF-8 encoding of ucs into buf and return the number of bytes written. Up to 4 bytes may be written. If you know that ucs is less than 0x10000 then at most 3 bytes will be written. If you wish to speed this up, remember that anything less than 0x80 is written as a single byte.

If ucs is greater than 0x10ffff this is an illegal character according to RFC 3629. These are converted as though they are 0xFFFD (REPLACEMENT CHARACTER).

RFC 3629 also says many other values for ucs are illegal (in the range 0xd800 to 0xdfff, or ending with 0xfffe or 0xffff). However I encode these as though they are legal, so that utf8encode/fl_utf8decode will be the identity for all codes between 0 and 0x10ffff.

11.11.3.25 fl_utf8from_mb()

```c
unsigned fl_utf8from_mb {
    char * dst,
    unsigned dstlen,
    const char * src,
    unsigned srclen }
```

Convert a filename from the locale-specific multibyte encoding used by Windows to UTF-8 as used by FLTK. Up to dstlen bytes are written to dst, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to dstlen then if you malloc a new array of...
size n+1 you will have the space needed for the entire string. If dstlen is zero then nothing is written and this call just measures the storage space needed.

On Unix or on Windows when a UTF-8 locale is in effect, this does not change the data. You may also want to check if fl_utf8test() returns non-zero, so that the filesystem can store filenames in UTF-8 encoding regardless of the locale.

### 11.11.3.26 fl_utf8froma()

```c
unsigned fl_utf8froma (  
    char * dst,  
    unsigned dstlen,  
    const char * src,  
    unsigned srclen )
```

Convert an ISO-8859-1 (ie normal c-string) byte stream to UTF-8.

It is possible this should convert Microsoft's CP1252 to UTF-8 instead. This would translate the codes in the range 0x80-0x9f to different characters. Currently it does not do this.

Up to dstlen bytes are written to dst, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to dstlen then if you malloc a new array of size n+1 you will have the space needed for the entire string. If dstlen is zero then nothing is written and this call just measures the storage space needed.

srclen is the number of bytes in src to convert.

If the return value equals srclen then this indicates that no conversion is necessary, as only ASCII characters are in the string.

### 11.11.3.27 fl_utf8fromwc()

```c
unsigned fl_utf8fromwc (  
    char * dst,  
    unsigned dstlen,  
    const wchar_t * src,  
    unsigned srclen )
```

Turn "wide characters" as returned by some system calls (especially on Windows) into UTF-8.

Up to dstlen bytes are written to dst, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to dstlen then if you malloc a new array of size n+1 you will have the space needed for the entire string. If dstlen is zero then nothing is written and this call just measures the storage space needed.

srclen is the number of words in src to convert. On Windows this is not necessarily the number of characters, due to there possibly being "surrogate pairs" in the UTF-16 encoding used. On Unix wchar_t is 32 bits and each location is a character.

On Unix if a src word is greater than 0x10ffff then this is an illegal character according to RFC 3629. These are converted as though they are 0xFFFD (REPLACEMENT CHARACTER). Characters in the range 0xd800 to 0xdfff, or ending with 0xfffe or 0xffff are also illegal according to RFC 3629. However I encode these as though they are legal, so that fl_utf8towc will return the original data.

On Windows "surrogate pairs" are converted to a single character and UTF-8 encoded (as 4 bytes). Mismatched halves of surrogate pairs are converted as though they are individual characters.

### 11.11.3.28 fl_utf8fwd()

```c
const char * fl_utf8fwd (  
    const char * p,  
    const char * start,  
    const char * end )
```

Move p forward until it points to the start of a UTF-8 character.

If it already points at the start of one then it is returned unchanged. Any UTF-8 errors are treated as though each byte of the error is an individual character.

start is the start of the string and is used to limit the backwards search for the start of a UTF-8 character. end is the end of the string and is assumed to be a break between characters. It is assumed to be greater than p.

This function is for moving a pointer that was jumped to the middle of a string, such as when doing a binary search for a position. You should use either this or fl_utf8back() depending on which direction your algorithm can handle
11.11 Unicode and UTF-8 functions

the pointer moving. Do not use this to scan strings, use fl_utf8decode() instead.

11.11.3.29 fl_utf8len()

```c
int fl_utf8len ( char c )
```

Returns the byte length of the UTF-8 sequence with first byte c, or -1 if c is not valid.
This function is helpful for finding faulty UTF-8 sequences.

See also

- fl_utf8len1

11.11.3.30 fl_utf8len1()

```c
int fl_utf8len1 ( char c )
```

Returns the byte length of the UTF-8 sequence with first byte c, or 1 if c is not valid.
This function can be used to scan faulty UTF-8 sequences, albeit ignoring invalid codes.

See also

- fl_utf8len

11.11.3.31 fl_utf8locale()

```c
int fl_utf8locale ( void )
```

Return true if the "locale" seems to indicate that UTF-8 encoding is used. If true the fl_utf8to_mb and fl_utf8from_mb don't do anything useful.

*It is highly recommended that you change your system so this does return true.* On Windows this is done by setting the "codepage" to CP_UTF8. On Unix this is done by setting $LC_CTYPE to a string containing the letters "utf" or "UTF" in it, or by deleting all $LC_∗ and $LANG environment variables. In the future it is likely that all non-Asian Unix systems will return true, due to the compatibility of UTF-8 with ISO-8859-1.

11.11.3.32 fl_utf8strlen()

```c
int fl_utf8strlen ( const char * text, int len )
```

Return the length in bytes of a UTF-8 string.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>text</code></td>
<td>encoded in UTF-8</td>
</tr>
<tr>
<td><code>len</code></td>
<td>number of Unicode characters, -1 to test until the end of text</td>
</tr>
</tbody>
</table>

Returns

number of bytes that make up the Unicode string

See also

- fl_utf_nb_char(const unsigned char *buf, int len)

11.11.3.33 fl_utf8test()

```c
int fl_utf8test ( const char * src, unsigned srclen )
```
Examines the first `src_len` bytes in `src` and returns a verdict on whether it is UTF-8 or not.

- Returns 0 if there is any illegal UTF-8 sequences, using the same rules as `fl_utf8decode()`. Note that some UCS values considered illegal by RFC 3629, such as 0xffff, are considered legal by this.
- Returns 1 if there are only single-byte characters (i.e., no bytes have the high bit set). This is legal UTF-8, but also indicates plain ASCII. It also returns 1 if `src_len` is zero.
- Returns 2 if there are only characters less than 0x800.
- Returns 3 if there are only characters less than 0x10000.
- Returns 4 if there are characters in the 0x10000 to 0x10ffff range.

Because there are many illegal sequences in UTF-8, it is almost impossible for a string in another encoding to be confused with UTF-8. This is very useful for transitioning Unix to UTF-8 filenames, you can simply test each filename with this to decide if it is UTF-8 or in the locale encoding. My hope is that if this is done we will be able to cleanly transition to a locale-less encoding.

### fl_utf8to_mb()

```c
unsigned fl_utf8to_mb (  
    const char * src,  
    unsigned src_len,  
    char * dst,  
    unsigned dst_len )
```

Convert the UTF-8 used by FLTK to the locale-specific encoding used for filenames (and sometimes used for data in files).

Unfortunately due to stupid design you will have to do this as needed for filenames. This is a bug on both Unix and Windows.

Up to `dst_len` bytes are written to `dst`, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to `dst_len` then if you malloc a new array of size `n+1` you will have the space needed for the entire string. If `dst_len` is zero then nothing is written and this call just measures the storage space needed.

If `fl_utf8locale()` returns true then this does not change the data.

### fl_utf8toa()

```c
unsigned fl_utf8toa (  
    const char * src,  
    unsigned src_len,  
    char * dst,  
    unsigned dst_len )
```

Convert a UTF-8 sequence into an array of 1-byte characters.

If the UTF-8 decodes to a character greater than 0xff then it is replaced with '?'.

Errors in the UTF-8 sequence are converted as individual bytes, same as `fl_utf8decoder()` does. This allows ISO-8859-1 text mistakenly identified as UTF-8 to be printed correctly (and possibly CP1252 on Windows).

Up to `dst_len` bytes are written to `dst`, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to `dst_len` then if you malloc a new array of size `n+1` you will have the space needed for the entire string. If `dst_len` is zero then nothing is written and this call just measures the storage space needed.

### fl_utf8toUtf16()

```c
unsigned fl_utf8toUtf16 (  
    const char * src,  
    unsigned src_len,  
    unsigned short * dst,  
    unsigned dst_len )
```

Convert a UTF-8 sequence into an array of 16-bit characters.
These are used by some system calls, especially on Windows.

**src** points at the UTF-8, and **srclen** is the number of bytes to convert.

**dst** points at an array to write, and **dstlen** is the number of locations in this array. At most **dstlen-1** words will be written there, plus a 0 terminating word. Thus this function will never overwrite the buffer and will always return a zero-terminated string. If **dstlen** is zero then **dst** can be null and no data is written, but the length is returned. The return value is the number of 16-bit words that would be written to **dst** if it were long enough, not counting the terminating zero. If the return value is greater or equal to **dstlen** it indicates truncation, you can then allocate a new array of size **return+1** and call this again.

Errors in the UTF-8 are converted as though each byte in the erroneous string is in the Microsoft CP1252 encoding. This allows ISO-8859-1 text mistakenly identified as UTF-8 to be printed correctly.

Unicode characters in the range 0x10000 to 0x10ffff are converted to "surrogate pairs" which take two words each (this is called UTF-16 encoding).

### 11.11.3.37 fl_utf8towc()

```c
unsigned fl_utf8towc (  
    const char * src,  
    unsigned srclen,  
    wchar_t * dst,  
    unsigned dstlen )  
```

Converts a UTF-8 string into a wide character string.

This function generates 32-bit wchar_t (e.g. "ucs4" as it were) except on Windows where it is equivalent to fl←utf8toUtf16 and returns UTF-16.

**src** points at the UTF-8, and **srclen** is the number of bytes to convert.

**dst** points at an array to write, and **dstlen** is the number of locations in this array. At most **dstlen-1** wchar_t will be written there, plus a 0 terminating wchar_t.

The return value is the number of wchar_t that would be written to **dst** if it were long enough, not counting the terminating zero. If the return value is greater or equal to **dstlen** it indicates truncation, you can then allocate a new array of size return+1 and call this again.

Notice that sizeof(wchar_t) is 2 on Windows and is 4 on Linux and most other systems. Where wchar_t is 16 bits, Unicode characters in the range 0x10000 to 0x10ffff are converted to "surrogate pairs" which take two words each (this is called UTF-16 encoding). If wchar_t is 32 bits this rather nasty problem is avoided.

Note that Windows includes Cygwin, i.e. compiled with Cygwin's POSIX layer (cygwin1.dll, –enable-cygwin), either native (GDI) or X11.

### 11.11.3.38 fl_utf_nb_char()

```c
int fl_utf_nb_char (  
    const unsigned char * buf,  
    int len )  
```

Returns the number of Unicode chars in the UTF-8 string.

See also

`fl_utf8strlen(const char *text, int len)`

### 11.11.3.39 fl_utf_strcasecmp()

```c
int fl_utf_strcasecmp (  
    const char * s1,  
    const char * s2 )  
```

UTF-8 aware strcasecmp - converts to Unicode and tests.

Returns

```
result of comparison
```

Return values

```
0 | if the strings are equal
```

Generated by Doxygen
Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if s1 is greater than s2</td>
</tr>
<tr>
<td>-1</td>
<td>if s1 is less than s2</td>
</tr>
</tbody>
</table>

11.11.3.40  fl_utf_strncasecmp()

```c
int fl_utf_strncasecmp(  
    const char * s1,  
    const char * s2,  
    int n)
```

UTF-8 aware strncasecmp - converts to lower case Unicode and tests.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>s1,s2</td>
<td>the UTF-8 strings to compare</td>
</tr>
<tr>
<td>n</td>
<td>the maximum number of UTF-8 characters to compare</td>
</tr>
</tbody>
</table>

Returns

result of comparison

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the strings are equal</td>
</tr>
<tr>
<td>&gt;0</td>
<td>if s1 is greater than s2</td>
</tr>
<tr>
<td>&lt;0</td>
<td>if s1 is less than s2</td>
</tr>
</tbody>
</table>

11.11.3.41  fl_utf_tolower()

```c
int fl_utf_tolower(  
    const unsigned char * str,  
    int len,  
    char * buf)
```

Converts the string str to its lower case equivalent into buf.  
Warning: to be safe buf length must be at least 3 * len [for 16-bit Unicode]

11.11.3.42  fl_utf_toupper()

```c
int fl_utf_toupper(  
    const unsigned char * str,  
    int len,  
    char * buf)
```

Converts the string str to its upper case equivalent into buf.  
Warning: to be safe buf length must be at least 3 * len [for 16-bit Unicode]

11.11.3.43  fl_wcwidth()

```c
int fl_wcwidth(  
    const char * src)
```

extended wrapper around fl_wcwidth_(unsigned int ucs) function.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>src</td>
</tr>
</tbody>
</table>
11.12 String handling functions

String handling functions declared in `<FL/fl_string_functions.h>`

**Functions**

- `char * fl_strdup (const char *s)`
  
  Cross platform interface to POSIX function `strdup()`.

11.12.1 Detailed Description

String handling functions declared in `<FL/fl_string_functions.h>`

11.12.2 Function Documentation

11.12.2.1 `fl_strdup()`

```c
char * fl_strdup (const char *s)
```

Cross platform interface to POSIX function `strdup()`.

The `fl_strdup()` function returns a pointer to a new string which is a duplicate of the string 's'. Memory for the new string is obtained with `malloc(3)`, and can be freed with `free(3)`.

Implementation:

- POSIX: `strdup()`
- WinAPI: `_strdup()`

11.13 Mac OS X-specific symbols

Mac OS X-specific symbols declared in `<FL/platform.H>`
Classes

- class Fl_Mac_App_Menu

Functions

- FLWindow * fl_mac_find (FLWindow *)
  
  Returns the FL_Window corresponding to the given macOS-specific window reference.

- CGContextRef fl_mac_gc ()
  
  Returns the macOS-specific graphics context for the current window.

- void fl_mac_set_about (Fl_Callback *cb, void *user_data, int shortcut=0)
  
  Attaches a callback to the "About myprog" item of the system application menu.

- FLWindow * fl_mac_xid (const Fl_Window *)
  
  Returns the macOS-specific window reference corresponding to the given FL_Window object.

- void fl_open_callback (void(*)(const char *))
  
  Register a function called for each file dropped onto an application icon.

Variables

- int fl_mac_os_version
  
  The version number of the running Mac OS X (e.g., 100604 for 10.6.4, 101300 for 10.13, 140102 for 14.1.2).

11.13.1 Detailed Description

Mac OS X-specific symbols declared in `<FL/platform.H>`

See also

The Apple OS X Interface

11.13.2 Function Documentation

11.13.2.1 fl_mac_set_about()

void fl_mac_set_about (Fl_Callback *cb, void *user_data, int shortcut = 0)

Attaches a callback to the "About myprog" item of the system application menu.

For back-compatibility. Equivalent to Fl_Sys_Menu_Bar::about(Fl_Callback *cb, void *user_data).

11.13.2.2 fl_open_callback()

void fl_open_callback (void(*)(const char *))

Register a function called for each file dropped onto an application icon.

This function is effective only on the Mac OS X platform. `cb` will be called with a single Unix-style file name and path. If multiple files were dropped, `cb` will be called multiple times.

This function should be called before `fl_open_display()` is called, either directly or indirectly (this happens at the first `show()` of a window), to be effective for files dropped on the application icon at launch time. It can also be called at any point to change the function used to open dropped files. A call with a NULL argument, after a previous call, makes the app ignore files dropped later.

11.13.3 Variable Documentation

11.13.3.1 fl_mac_os_version

int fl_mac_os_version [extern]

The version number of the running Mac OS X (e.g., 100604 for 10.6.4, 101300 for 10.13, 140102 for 14.1.2).

FLTK initializes this global variable before main() begins running. If the value is needed in a static initializer, a previous call to `Fl::system_driver()` makes sure `fl_mac_os_version` has been initialized.
11.14 Common Dialog Classes and Functions

Common dialog functions for file selection, message output, and more.

Files

- file fl_ask.cxx
  Utility functions for common dialogs.

Classes

- class Fl_Color_Chooser
  The Fl_Color_Chooser widget provides a standard RGB color chooser.

- class Fl_File_Chooser
  The Fl_File_Chooser widget displays a standard file selection dialog that supports various selection modes.

Functions

- void fl_alert (const char *fmt,...)
  Shows an alert message dialog box.

- int fl_ask (const char *fmt,...)
  Shows a dialog displaying the fmt message, this dialog features 2 yes/no buttons.

- void fl_beep (int type)
  Emits a system beep.

- int fl_choice (const char *fmt, const char *b0, const char *b1, const char *b2,...)
  Shows a dialog displaying the printf style fmt message.

- int fl_choice_n (const char *fmt, const char *b0, const char *b1, const char *b2,...)
  Shows a dialog displaying the printf style fmt message.

- int fl_color_chooser (const char *name, double &r, double &g, double &b, int cmode)
  Pops up a window to let the user pick an arbitrary RGB color.

- int fl_color_chooser (const char *name, uchar &r, uchar &g, uchar &b, int cmode)
  Pops up a window to let the user pick an arbitrary RGB color.

- char * fl_dir_chooser (const char *message, const char *fname, int relative)
  Shows a file chooser dialog and gets a directory.

- char * fl_file_chooser (const char *message, const char *pat, const char *fname, int relative)
  Shows a file chooser dialog and gets a filename.

- void fl_file_chooser_callback (void(*)(const char *))
  Set the file chooser callback.

- void fl_file_chooser_ok_label (const char *l)
  Set the "OK" button label.

- const char * fl_input (const char *fmt, const char *defstr,...)
  Shows an input dialog displaying the fmt message with variable arguments.

- const char * fl_input (int maxchar, const char *fmt, const char *defstr,...)
  Shows an input dialog displaying the fmt message with variable arguments.

- void fl_message (const char *fmt,...)
  Shows an information message dialog box.

- int fl_message_hotspot ()
  Gets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.

- void fl_message_hotspot (int enable)
  Sets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.
Gets the Fl_Box icon container of the current default dialog used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

- **void fl_message_icon_label (const char *str)**
  Sets the icon label of the dialog window used in many common dialogs.

- **void fl_message_position (const int x, const int y, const int center)**
  Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

- **void fl_message_position (Fl_Widget *widget)**
  Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

- **int fl_message_position (int *x, int *y)**
  Gets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

- **void fl_message_title (const char *title)**
  Sets the title of the dialog window used in many common dialogs.

- **void fl_message_title_default (const char *title)**
  Sets the default title of the dialog window used in many common dialogs.

- **const char * fl_password (const char *fmt, const char *defstr,...)**
  Shows an input dialog displaying the fmt message with variable arguments.

- **const char * fl_password (int maxchar, const char *fmt, const char *defstr,...)**
  Shows an input dialog displaying the fmt message with variable arguments.

**Variables**

- **static void ( Fl::error ) (const char *,...) = Fl_System_Driver::error**
  FLTKit calls Fl::error() to output a normal error message.

- **static void ( Fl::fatal ) (const char *,...) = Fl_System_Driver::fatal**
  FLTKit calls Fl::fatal() to output a fatal error message.

- **const char * fl_cancel = "Cancel"**
  string pointer used in common dialogs, you can change it to another language

- **const char * fl_close = "Close"**
  string pointer used in common dialogs, you can change it to another language

- **Fl_Font fl_message_font_ = FL_HELVETICA**

- **Fl_Fontsize fl_message_size_ = -1**

- **const char * fl_no = "No"**
  string pointer used in common dialogs, you can change it to another language

- **const char * fl_ok = "OK"**
  string pointer used in common dialogs, you can change it to another language

- **const char * fl_yes = "Yes"**
  string pointer used in common dialogs, you can change it to another language

- **static void ( Fl::warning ) (const char *,...) = Fl_System_Driver::warning**
  FLTKit calls Fl::warning() to output a warning message.

### 11.14.1 Detailed Description

Common dialog functions for file selection, message output, and more.

### 11.14.2 Function Documentation

#### 11.14.2.1 fl_alert()

```cpp
void fl_alert ( 
    const char * fmt,
    ... 
)
```

Shows an alert message dialog box.

#include <FL/fl_ask.h>
11.14 Common Dialog Classes and Functions

Parameters

| in  | fmt | can be used as a sprintf-like format and variables for the message text |

11.14.2.2 fl_ask()

```c
int fl_ask (const char * fmt, ... )
```

Shows a dialog displaying the `fmt` message, this dialog features 2 yes/no buttons.

```c
#include <FL/fl_ask.H>
```

Parameters

| in  | fmt | can be used as a sprintf-like format and variables for the message text |

Return values

| 0   | if the no button is selected |
| 1   | if yes is selected |

**Deprecated** fl_ask() is deprecated since it uses "Yes" and "No" for the buttons which does not conform to the current FLTK Human Interface Guidelines. Use fl_choice() with the appropriate verbs instead.

11.14.2.3 fl_beep()

```c
void fl_beep ( int type )
```

Emits a system beep.

This function is platform specific. Depending on the input `type` a different sound may be played or the system speaker may beep with a different volume.

On X the system speaker is used which may not work at all on newer systems that don't have a speaker. Since 1.4.0 FL_BEEP_DEFAULT and other types honor the system or user settings whereas FL_BEEP_ERROR uses 100% volume. This may be changed in a future version.

On Wayland an ASCII BEL (0x07) is output to stderr.

On Windows the MessageBeep() function is used to play different sounds depending on the `type` argument.

On macOS the system beep function NSBeep() is used for FL_BEEP_DEFAULT and FL_BEEP_ERROR. Other types are ignored.

On other platforms the behavior is undefined and may change in the future.

Parameters

| in  | type | The beep type from the Fl_Beep enumeration (optional) |

```c
#include <FL/fl_ask.H>
```

11.14.2.4 fl_choice()

```c
int fl_choice (const char * fmt, const char * b0, const char * b1, const char * b2, ... )
```

Shows a dialog displaying the printf style `fmt` message.

Generated by Doxygen
This dialog features up to 3 customizable choice buttons which are specified in order of right-to-left in the dialog, e.g.

```c
#include <FL/fl_ask.H>

Three choices with printf() style formatting:

```c
int num_msgs = GetNumberOfMessages();
switch ( fl_choice("What to do with %d messages?", "Send", "Save", "Delete", num_msgs) ) {
  case 0: .. // Send
  case 1: .. // Save (default)
  case 2: .. // Delete
  ...
}
```

Three choice example:

```c
switch ( fl_choice("How many bedrooms?", "Zero", "One", "Two") ) {
  case 0: .. // "Zero"
  case 1: .. // "One" (default)
  case 2: .. // "Two"
}
```

Two choice example:

```c
switch ( fl_choice("Empty trash?", "Yes", "No", 0) ) {
  case 0: .. // Yes
  case 1: .. // No (default)
}
```

One choice example:

```c
fl_choice("All hope is lost.", "OK", 0, 0); // "OK" default
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fmt</code></td>
<td>can be used as an sprintf-like format and variables for the message text</td>
</tr>
<tr>
<td><code>b0</code></td>
<td>text label for right button 0</td>
</tr>
<tr>
<td><code>b1</code></td>
<td>text label for middle button 1 (can be 0)</td>
</tr>
<tr>
<td><code>b2</code></td>
<td>text label for left button 2 (can be 0)</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the button with b0 text is pushed or the user pressed the Escape key or clicked the window close button</td>
</tr>
<tr>
<td>1</td>
<td>if the button with b1 text is pushed or the user pressed the Return key</td>
</tr>
<tr>
<td>2</td>
<td>if the button with b2 text is pushed</td>
</tr>
</tbody>
</table>

### 11.14.2.5 fl_choice_n()

```c
int fl_choice_n (const char *fmt, const char *b0, const char *b1, const char *b2, ...)
```

Shows a dialog displaying the printf style `fmt` message. This function is like `fl_choice()` but returns -1 if the dialog window was closed by pressing the Escape key or the window close button rather than pushing one of the dialog buttons.

See also

`fl_choice()`

#### Parameters

<table>
<thead>
<tr>
<th>In</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fmt</code></td>
<td>can be used as a sprintf-like format and variables for the message text</td>
</tr>
<tr>
<td><code>b0</code></td>
<td>text label for right button 0</td>
</tr>
<tr>
<td><code>b1</code></td>
<td>text label for middle button 1 (can be 0)</td>
</tr>
<tr>
<td><code>b2</code></td>
<td>text label for left button 2 (can be 0)</td>
</tr>
</tbody>
</table>

#### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>reserved, FLTK 1.3 only: another dialog is still open (not possible in 1.4)</td>
</tr>
<tr>
<td>-2</td>
<td>if the dialog was closed by pushing the window close button</td>
</tr>
<tr>
<td>-1</td>
<td>if the dialog was closed by hitting Escape</td>
</tr>
<tr>
<td>0</td>
<td>if the button with b0 text is pushed</td>
</tr>
<tr>
<td>1</td>
<td>if the button with b1 text is pushed</td>
</tr>
<tr>
<td>2</td>
<td>if the button with b2 text is pushed</td>
</tr>
</tbody>
</table>

### 11.14.2.6 fl_color_chooser() [1/2]

```c
int fl_color_chooser (const char *name, double & r, double & g, double & b, int cmode ) [related]
```

Pops up a window to let the user pick an arbitrary RGB color.
Note

\#include <FL/Fl_Color_Chooser.H>

Figure 11.10 fl_color_chooser

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>Title label for the window</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>r,g,b</td>
<td>Color components in the range 0.0 to 1.0.</td>
</tr>
<tr>
<td>in</td>
<td>cmode</td>
<td>Optional mode for color chooser. See mode(int). Default -1 if none (rgb mode).</td>
</tr>
</tbody>
</table>

Return values

| 1 | if user confirms the selection |
| 0 | if user cancels the dialog |

11.14.2.7 fl_color_chooser() [2/2]

```c
int fl_color_chooser (  
    const char * name,  
    uchar & r,  
    uchar & g,  
    uchar & b,  
    int cmode ) [related]
```

Pops up a window to let the user pick an arbitrary RGB color.
11.14 Common Dialog Classes and Functions

Note

```c
#include <FL/Fl_Color_Chooser.H>
```

![Figure 11.11 fl_color_chooser](image)

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>Title label for the window</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>r,g,b</td>
<td>Color components in the range 0 to 255.</td>
</tr>
<tr>
<td>in</td>
<td>cmode</td>
<td>Optional mode for color chooser. See <code>mode(int)</code>. Default -1 if none (rgb mode).</td>
</tr>
</tbody>
</table>

**Return values**

| 1  | if user confirms the selection |
| 0  | if user cancels the dialog |

11.14.2.8  

```c
char * fl_dir_chooser ( const char * message, const char * fname, int relative ) [related]
```

Shows a file chooser dialog and gets a directory.

Note

```c
#include <FL/Fl_File_Chooser.H>
```

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>message</th>
<th>title bar text</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fname</td>
<td>initial/default directory name</td>
</tr>
<tr>
<td>in</td>
<td>relative</td>
<td>0 for absolute path return, relative otherwise</td>
</tr>
</tbody>
</table>
Returns

the directory path string chosen by the user or NULL if user cancels.

11.14.2.9 fl_file_chooser()

char * fl_file_chooser (
    const char * message,
    const char * pat,
    const char * fname,
    int relative ) [related]

Shows a file chooser dialog and gets a filename.

Note

#include <FL/Fl_File_Chooser.H>

![Fl_File_Chooser Test](image)

Figure 11.12 Fl_File_Chooser

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in message</td>
<td>text in title bar</td>
</tr>
<tr>
<td>in pat</td>
<td>filename pattern filter</td>
</tr>
<tr>
<td>in fname</td>
<td>initial/default filename selection</td>
</tr>
<tr>
<td>in relative</td>
<td>0 for absolute path name, relative path name otherwise</td>
</tr>
</tbody>
</table>
Returns

the user selected filename, in absolute or relative format or NULL if user cancels

### 11.14.2.10 fl_file_chooser_callback()

```c
void fl_file_chooser_callback (  
  void(*)(const char *) cb ) [related]
```

Set the file chooser callback.

**Note**

```
#include <FL/Fl_File_Chooser.H>
```

### 11.14.2.11 fl_file_chooser_ok_label()

```c
void fl_file_chooser_ok_label (  
  const char * l ) [related]
```

Set the "OK" button label.

**Note**

```
#include <FL/Fl_File_Chooser.H>
```

### 11.14.2.12 fl_input() [1/2]

```c
const char * fl_input (  
  const char * fmt,  
  const char * defstr,  
  ... )
```

Shows an input dialog displaying the `fmt` message with variable arguments.

Returns the string in an internally allocated buffer that may be changed later. You **must** copy the string immediately after return from this method - at least before the next execution of the event loop.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fmt</code></td>
<td>can be used as a sprintf-like format and variables for the message text</td>
</tr>
<tr>
<td><code>defstr</code></td>
<td>defines the default returned string if no text is entered</td>
</tr>
</tbody>
</table>

Returns

the user string input if OK was pushed

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NULL</code></td>
<td>if Cancel was pushed or the window was closed by the user</td>
</tr>
</tbody>
</table>

### 11.14.2.13 fl_input() [2/2]

```c
const char * fl_input (  
  int maxchar,  
  const char * fmt,  
  const char * defstr,  
  ... )
```

Shows an input dialog displaying the `fmt` message with variable arguments.

This is the same as `const char * fl_input(const char *fmt, const char *defstr, ...)` except that it has an additional...
parameter to limit the number of characters the user can input. Returns the string in an internally allocated buffer that may be changed later. You must copy the string immediately after return from this method - at least before the next execution of the event loop.

```c
#include <FL/fl_ask.H>
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>maxchar</code></td>
<td>maximum number of characters the user can input (UTF-8 aware)</td>
</tr>
<tr>
<td><code>fmt</code></td>
<td>can be used as a sprintf-like format and variables for the message text</td>
</tr>
<tr>
<td><code>defstr</code></td>
<td>defines the default returned string if no text is entered</td>
</tr>
</tbody>
</table>

### Returns

the user string input if OK was pushed

### Return values

- `NULL` if Cancel was pushed or the window was closed by the user

---

### 11.14.2.14 fl_message()

```c
void fl_message (const char * fmt, ... )
```

Shows an information message dialog box.

```c
#include <FL/fl_ask.H>
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fmt</code></td>
<td>can be used as a sprintf-like format and variables for the message text</td>
</tr>
</tbody>
</table>

---

### 11.14.2.15 fl_message_hotspot() [1/2]

```c
int fl_message_hotspot { void }
```

Gets whether or not to move the message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()` to follow the mouse pointer.

This is a permanent setting. It remains active and affects the window position unless overridden by an explicit positioning request by means of one of the `fl_message_position()` variants.

```c
#include <FL/fl_ask.H>
```

### Returns

- `0` if disabled, non-zero otherwise

### See also

- `void fl_message_hotspot(int)`
- `int fl_message_position(int *x, int *y)`
- `void fl_message_position(Fl_Widget *)`
- `fl_message_position()`

---

### 11.14.2.16 fl_message_hotspot() [2/2]

```c
void fl_message_hotspot (int enable)
```

---

Generated by Doxygen
Sets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer. The default is enabled, so that the default button is the hotspot and appears at the mouse position.

```c
#include <FL/fl_ask.H>
```

### Parameters

| enabled | non-zero enables hotspot behavior, 0 disables hotspot |

#### 11.14.2.17 fl_message_icon()

```c
FL_Widget * fl_message_icon ( )
```

Gets the Fl_Box icon container of the current default dialog used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

The return value cannot be Null. The object pointed to is an Fl_Box widget. The returned pointer (Fl_Widget *) can be safely cast to an Fl_Box pointer.

**Note**

You can set some attributes of this default icon box. These attributes are sticky, i.e. they will be used in all subsequent common dialogs unless overridden by specific "one shot" variables. Setting any attribute except those mentioned below causes undefined behavior.

**Supported icon attributes:**

- box()
- labelfont()
- labelsize()
- color()
- labelcolor()
- image()
- align()

The icon size can not be changed. If you set an image() you should scale it to the available size, i.e. w() and h() of the icon box.

```c
#include <FL/fl_ask.H>
```

#### 11.14.2.18 fl_message_icon_label()

```c
void fl_message_icon_label ( const char * str )
```

Sets the icon label of the dialog window used in many common dialogs. This icon label will be used in the next call of one of the common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

The label str is stored internally as a reference, it must be in scope until the dialog function (e.g. fl_choice) is called.

It applies only to the next call of one of the common dialogs and will be reset after that call so the next dialog will use its default label unless set again.

**Note**

This label string must be short, usually only one character so it fits in the icon box. You can use any valid UTF-8 character, e.g. the Euro sign ("€") which is three bytes in UTF-8 encoding.

```c
#include <FL/fl_ask.H>
```
11.14.2.19  fl_message_position()  [1/3]

void fl_message_position (const int x, const int y, const int center);

Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

The position set with this method overrides the hotspot setting, i.e. setting a position has higher priority than the hotspot mode set by fl_message_hotspot(int).

The preferred position set by any of the fl_message_position() variants affects only the next call of one of the common dialogs. The preferred position is reset to 0 (unset) as soon as the dialog is shown.

If the optional argument center is non-zero (true) the message box will be centered at the given coordinates rather than using the X/Y position as the window position (top left corner).

See also

int fl_message_position(int *x, int *y)

11.14.2.20  fl_message_position()  [2/3]

void fl_message_position (Fl_Widget *widget);

Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

The message box will be centered over the given widget or window extensions.

Everything else is like fl_message_position(int, int, int) with argument ‘center’ set to 1.

See also

int fl_message_position(int x, int y, int center)

11.14.2.21  fl_message_position()  [3/3]

int fl_message_position (int *x, int *y);

Gets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

# include <FL/fl_ask.H>
The position set with this method overrides the hotspot setting, i.e. setting a position has higher priority than the hotspot mode set by `fl_message_hotspot(int)`.
The preferred position set by any of the `fl_message_position()` variants affects only the next call of one of the common dialogs. The preferred position is reset to 0 (unset) as soon as the dialog is shown.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>x</th>
<th>Preferred X position, returns -1 if not set</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>Preferred Y position, returns -1 if not set</td>
</tr>
</tbody>
</table>

Returns

whether position is currently set or not

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>position is not set (hotspot may be enabled or not)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>position is set (window position)</td>
</tr>
<tr>
<td>2</td>
<td>position is set (message box centered)</td>
</tr>
</tbody>
</table>

See also

`fl_message_hotspot()`
`fl_message_hotspot(int)`
`fl_message_position(int, int)`
`fl_message_position(const int x, const int y, const int center)`
`fl_message_position(Fl_Widget *)`

### 11.14.2.22 fl_message_title()

```
void fl_message_title (const char *title)
```

Sets the title of the dialog window used in many common dialogs.

This window title will be used in the next call of one of the common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`.

The title string is copied internally, so that you can use a local variable or free the string immediately after this call. It applies only to the next call of one of the common dialogs and will be reset to an empty title (the default for all dialogs) after that call.

```
#include <FL/fl_ask.H>
```

Parameters

| in | title | window label, string copied internally |

### 11.14.2.23 fl_message_title_default()

```
void fl_message_title_default (const char *title)
```

Sets the default title of the dialog window used in many common dialogs.

This window title will be used in all subsequent calls of one of the common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`, unless a specific title has been set with `fl_message_title(const char *title)`.

The default is no title. You can override the default title for a single dialog with `fl_message_title(const char *title)`.

The title string is copied internally, so that you can use a local variable or free the string immediately after this call.
#include <FL/fl_ask.H>

## Parameters

**in**  
`title`  
default window label, string copied internally

### 11.14.24  `fl_password()` [1/2]

```c
const char * fl_password {
    const char * fmt,
    const char * defstr,
    ... }
```

Shows an input dialog displaying the `fmt` message with variable arguments.  
Like `fl_input()` except the input text is not shown, "*" or similar replacement characters are displayed instead.

```c
#include <FL/fl_ask.H>
```

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fmt</code></td>
<td>can be used as an <code>sprintf</code>-like format and variables for the message text</td>
</tr>
<tr>
<td><code>defstr</code></td>
<td>defines the default returned string if no text is entered</td>
</tr>
</tbody>
</table>

### Returns

the user string input if OK was pushed

### Return values

`NULL` if Cancel was pushed or the window was closed by the user

### 11.14.25  `fl_password()` [2/2]

```c
const char * fl_password {
    int maxchar,
    const char * fmt,
    const char * defstr,
    ... }
```

Shows an input dialog displaying the `fmt` message with variable arguments.  
Like `fl_input()` except the input text is not shown, "*" or similar replacement characters are displayed instead.

```c
#include <FL/fl_ask.H>
```

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>maxchar</code></td>
<td>input length limit in chars, 0 = no limit</td>
</tr>
<tr>
<td><code>fmt</code></td>
<td>can be used as an <code>sprintf</code>-like format and variables for the message text</td>
</tr>
<tr>
<td><code>defstr</code></td>
<td>defines the default returned string if no text is entered</td>
</tr>
</tbody>
</table>

### Returns

the user string input if OK was pushed

### Return values

`NULL` if Cancel was pushed or the window was closed by the user
11.14.3  Variable Documentation

11.14.3.1  error

void(* Fl::error)(const char *format,...) = Fl_System_Driver::error [static]

FTLK calls Fl::error() to output a normal error message. The default version on
Windows displays the error message in a MessageBox window. The default version on all
other platforms prints the error message to stderr. You can override the behavior by setting
the function pointer to your own routine. Fl::error() means there is a recoverable error
such as the inability to read an image file. The default implementation returns after
displaying the message.

Note

#include <FL/Fl.H>

11.14.3.2  fatal

void(* Fl::fatal)(const char *format,...) = Fl_System_Driver::fatal [static]

FTLK calls Fl::fatal() to output a fatal error message. The default version on Windows
displays the error message in a MessageBox window. The default version on all other
platforms prints the error message to stderr. You can override the behavior by setting
the function pointer to your own routine. Fl::fatal() must not return, as FTLK is in an
unusable state, however your version may be able to use longjmp or an exception
to continue, as long as it does not call FTLK again. The default implementation exits with status 1 after
displaying the message.

Note

#include <FL/Fl.H>

11.14.3.3  warning

void(* Fl::warning)(const char *format,...) = Fl_System_Driver::warning [static]

FTLK calls Fl::warning() to output a warning message. The default version on Windows returns
without printing a warning message, because Windows programs normally
don't have stderr (a console window) enabled. The default version on all other platforms
prints the warning message to stderr. You can override the behavior by setting the function pointer
to your own routine. Fl::warning() means that there was a recoverable problem, the display may be messed up, but the user can probably
keep working - all X protocol errors call this, for example. The default implementation returns after displaying
the message.

Note

#include <FL/Fl.H>

11.15  The gtk_shell1 interface

gtk_shell is a protocol extension providing additional features for clients implementing it.

Classes

• struct gtk_shell1_listener

Macros

• #define GTK_SHELL1_CAPABILITIES_SINCE_VERSION 1
• #define GTK_SHELL1_GET_GTK_SURFACE_SINCE_VERSION 1
• #define GTK_SHELL1_NOTIFY_LAUNCH_SINCE_VERSION 3
• #define GTK_SHELL1_SET_STARTUP_ID_SINCE_VERSION 1
• #define GTK_SHELL1_SYSTEM_BELL_SINCE_VERSION 1
11.15.1 Detailed Description

gtk_shell is a protocol extension providing additional features for clients implementing it.

11.16 The gtk_surface1 interface

Classes

• struct gtk_surface1_listener

Macros

• #define GTK_SURFACE1_CONFIGURE_EDGES_SINCE_VERSION 2
• #define GTK_SURFACE1_CONFIGURE_SINCE_VERSION 1
• #define GTK_SURFACE1_PRESENT_SINCE_VERSION 1
• #define GTK_SURFACE1_RELEASE_SINCE_VERSION 4
• #define GTK_SURFACE1_REQUEST_FOCUS_SINCE_VERSION 3
• #define GTK_SURFACE1_SET_DBUS_PROPERTIES_SINCE_VERSION 1
• #define GTK_SURFACE1_SET_MODAL_SINCE_VERSION 1
• #define GTK_SURFACE1_STATE_TILED_BOTTOM_SINCE_VERSION 2
• #define GTK_SURFACE1_STATE_TILED_LEFT_SINCE_VERSION 2
• #define GTK_SURFACE1_STATE_TILED_RIGHT_SINCE_VERSION 2
• #define GTK_SURFACE1_STATE_TILED_TOP_SINCE_VERSION 2
• #define GTK_SURFACE1_TITLEBAR_GESTURE_SINCE_VERSION 5
• #define GTK_SURFACE1_UNSET_MODAL_SINCE_VERSION 1

11.16.1 Detailed Description

11.17 File names and URI utility functions

File names and URI functions defined in `<FL/filename.H>`

Macros

• #define FL_PATH_MAX 2048
  all path buffers should use this length

Typedefs

• typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)
  File sorting function.

Functions

• void fl_decode_uri (char *uri)
  Decodes a URL-encoded string.
• int fl_filename_absolute (char *to, int tolen, const char *from)
  Makes a filename absolute from a relative filename to the current working directory.
• int fl_filename_absolute (char *to, int tolen, const char *from, const char *cwd)
  Concatenate the absolute path base with from to form the new absolute path in to.
• int fl_filename_expand (char *to, int tolen, const char *from)
  Expands a filename containing shell variables and tilde (~).
• const char * fl_filename_ext (const char *buf)
  Gets the extension of a filename.
• void fl_filename_free_list (struct dirent ***, int n)
Free the list of filenames that is generated by \texttt{fl_filename_list(\)}.  

- \texttt{int fl_filename_isdir (const char ∗name)}
  Determines if a file exists and is a directory from its filename.
- \texttt{int fl_filename_list (const char ∗d, struct dirent **∗∗l, Fl_File_Sort_F ∗s=fl_numericsort)}
  Portable and const-correct wrapper for the \texttt{scandir()} function.
- \texttt{int fl_filename_match (const char ∗name, const char ∗∗pattern)}
  Checks if a string \(s\) matches a pattern \(p\).
- \texttt{const char ∗ fl_filename_name (const char ∗filename)}
  Gets the file name from a path.
- \texttt{int fl_filename_relative (char ∗to, int tolen, const char ∗from)}
  Makes a filename relative to the current working directory.
- \texttt{int fl_filename_relative (char ∗to, int tolen, const char ∗from, const char ∗cwd)}
  Makes a filename relative to any other directory.
- \texttt{char ∗ fl_filename_setext (char ∗to, int tolen, const char ∗ext)}
  Replaces the extension in \(buf\) of \(max\).
- \texttt{int fl_open_uri (const char ∗uri, char ∗msg, int msglen)}
  Opens the specified Uniform Resource Identifier (URI).

### 11.17.1 Detailed Description

File names and URI functions defined in <\texttt{FL/filename.H}>.

### 11.17.2 Typedef Documentation

#### 11.17.2.1 \texttt{Fl_File_Sort_F}

typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)

File sorting function.

See also

\texttt{fl_filename_list(\)}

### 11.17.3 Function Documentation

#### 11.17.3.1 \texttt{fl_decode_uri()}

\texttt{void fl_decode_uri (char ∗uri)}

Decodes a URL-encoded string.

In a Uniform Resource Identifier (URI), all non-ASCII bytes and several others (e.g., '<', ',', ') are URL-encoded using 3 bytes by "%XY" where XY is the hexadecimal value of the byte. This function decodes the URI restoring its original UTF-8 encoded content. Decoding is done in-place.

#### 11.17.3.2 \texttt{fl_filename_absolute() [1/2]}

\texttt{int fl_filename_absolute (char ∗to, int tolen, const char ∗from)}

Makes a filename absolute from a relative filename to the current working directory.

```c
#include <FL/filename.H>
[..]
fl_chdir("/var/tmp");
fl_filename_absolute(out, sizeof(out), ".foo.txt"); // out="/var/tmp/foo.txt"
f1_filename_absolute(out, sizeof(out), ".foo.txt"); // out="/var/tmp/foo.txt"
fl_filename_absolute(out, sizeof(out), ".log/messages"); // out="/var/log/messages"
```
380

Topic Documentation

Parameters

out
in
in

to

resulting absolute filename

tolen
from

size of the absolute filename buffer
relative filename

Returns
0 if no change, non zero otherwise

11.17.3.3

fl_filename_absolute() [2/2]

int fl_filename_absolute (
char ∗ to,
int tolen,
const char ∗ from,
const char ∗ base )

Concatenate the absolute path base with from to form the new absolute path in to.
#include <FL/filename.H>
char out[FL_PATH_MAX];
fl_filename_absolute(out, sizeof(out), "../foo.txt", "/var/tmp");
// out="/var/foo.txt"
fl_filename_absolute(out, sizeof(out), "../local/bin", "/usr/bin"); // out="/usr/local/bin"

Parameters

out
in
in
in

to

resulting absolute filename

tolen
from
base

size of the absolute filename buffer
relative filename
from is relative to this absolute file path

Returns
0 if no change, non zero otherwise

11.17.3.4

fl_filename_expand()

int fl_filename_expand (
char ∗ to,
int tolen,
const char ∗ from )

Expands a filename containing shell variables and tilde (∼).
Currently handles these variants:
"~username"
"~/file"
"$VARNAME"

// if ’username’ does not exist, result will be unchanged
// does NOT handle ${VARNAME}

Examples:
#include <FL/filename.H>
[..]
putenv("TMPDIR=/var/tmp");
fl_filename_expand(out, sizeof(out), "~fred/.cshrc");
fl_filename_expand(out, sizeof(out), "~/.cshrc");
fl_filename_expand(out, sizeof(out), "$TMPDIR/foo.txt");

// out="/usr/fred/.cshrc"
// out="/usr/<yourname>/.cshrc"
// out="/var/tmp/foo.txt"

Parameters

out
in
in

to

resulting expanded filename

tolen

size of the expanded filename buffer

from

filename containing shell variables

Generated by Doxygen


11.17 File names and URI utility functions

11.17.3.5  fl_filename_ext()

const char * fl_filename_ext (  
    const char * buf )

Gets the extension of a filename.

```c
#include <FL/filename.H>
const char * out;
out = fl_filename_ext("/some/path/foo.txt"); // result: ".txt"
out = fl_filename_ext("/some/path/foo"); // result: NULL
```

Parameters

- `buf` the filename to be parsed

Returns

- a pointer to the extension (including ".") if any or NULL otherwise

11.17.3.6  fl_filename_free_list()

void fl_filename_free_list (  
    struct dirent *** list,  
    int n )

Free the list of filenames that is generated by fl_filename_list().
Free everything that was allocated by a previous call to fl_filename_list(). Use the return values as parameters for this function.

Parameters

- `list` table containing the resulting directory listing
- `n` number of entries in the list

11.17.3.7  fl_filename_isdir()

int fl_filename_isdir (  
    const char * n )

Determines if a file exists and is a directory from its filename.

```c
#include <FL/filename.H>
fl_filename_isdir("/etc"); // returns non-zero
fl_filename_isdir("/etc/hosts"); // returns 0
```

Parameters

- `n` the filename to parse

Returns

- non zero if file exists and is a directory, zero otherwise

11.17.3.8  fl_filename_list()

int fl_filename_list (  
    const char * d,  
    const char * buf )

Returns

- 0 if no change, non zero otherwise
dirent *** list,
    Fl_File_Sort_F * sort )

Portable and const-correct wrapper for the scandir() function.
For each file in that directory a "dirent" structure is created. The only portable thing about a dirent is that dirent.
    d_name is the nul-terminated file name. A pointers array to these dirent's is created and a pointer to the array is
    returned in list. The number of entries is given as a return value. If there is an error reading the directory a number
    less than zero is returned, and errno has the reason; errno does not work under Windows.

Include:
    #include <FL/filename.H>

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>d</th>
<th>the name of the directory to list. It does not matter if it has a trailing slash.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>list</td>
<td>table containing the resulting directory listing</td>
</tr>
<tr>
<td>in</td>
<td>sort</td>
<td>sorting functor:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fl_alphasort: The files are sorted in ascending alphabetical order; upper and lowercase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>letters are compared according to their ASCII ordering uppercase before lowercase.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fl_casealphasort: The files are sorted in ascending alphabetical order; upper and lowercase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>letters are compared equally case is not significant.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fl_casenumeric sorter: The files are sorted in ascending &quot;alphanumeric&quot; order, where an</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attempt is made to put unpadded numbers in consecutive order; upper and lowercase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>letters are compared equally case is not significant.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fl_numeric sorter: The files are sorted in ascending &quot;alphanumeric&quot; order, where an</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attempt is made to put unpadded numbers in consecutive order; upper and lowercase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>letters are compared according to their ASCII ordering - uppercase before lowercase.</td>
</tr>
</tbody>
</table>

Returns
    the number of entries if no error, a negative value otherwise.

Todo should support returning OS error messages

11.17.3.9 fl_filename_match()

int fl_filename_match (  
    const char * s,  
    const char * p  
)

Checks if a string s matches a pattern p.
The following syntax is used for the pattern:

• * matches any sequence of 0 or more characters.
• ? matches any single character.
• [set] matches any character in the set. Set can contain any single characters, or a-z to represent a range. To match ] or - they must be the first characters. To match ∧ or ! they must not be the first characters.
• [^set] or [iset] matches any character not in the set.
• {X|Y|Z} or (X,Y,Z) matches any one of the subexpressions literally.
• \x quotes the character x so it has no special meaning.
• x all other characters are matched "exactly" on a case-insensitive basis.

Notes:

• s and p are matched on a char/byte basis, not as UCS codepoints or UTF-8 sequences.
• [set] ranges must run from low to high, i.e. [a-z] and not [z-a]
• [set] comparison is case-sensitive, i.e. [a-z] won't match "A".
• \x only applies to the fl_filename_match special characters * [ 

Include:
#include <FL/filename.H>

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>s</td>
</tr>
<tr>
<td>in</td>
<td>p</td>
</tr>
</tbody>
</table>

Returns

non zero if the string matches the pattern

11.17.3.10 fl_filename_name()

c const char * fl_filename_name (  
  const char * filename )

Gets the file name from a path.
Similar to basename(3), exceptions shown below.
#include <FL/filename.H>  
[.]  
const char *out;  
out = fl_filename_name("/usr/lib"); // out="lib"  
out = fl_filename_name("/usr/"); // out="" (basename(3) returns "usr" instead)  
out = fl_filename_name("/"); // out="" (basename(3) returns "/" instead)  
out = fl_filename_name("."); // out="."  
out = fl_filename_name(".."); // out=".."  

Returns

a pointer to the char after the last slash, or to filename if there is none.

11.17.3.11 fl_filename_relative() [1/2]

int fl_filename_relative (  
  char * to,  
  int tolen,  
  const char * from )

Makes a filename relative to the current working directory.
Return the from path made relative to the working directory, similar to C++17 std::filesystem::path←::lexically_relative. This function can also be called with a fourth argument for a user supplied base directory path.
These conversions are purely lexical. They do not check that the paths exist, do not follow symlinks, and do not access the filesystem at all.
Path arguments must be absolute (start at the root directory) and must not contain . or .. segments, or double separators. A single trailing separator is ok.
On Windows, path arguments must start with a drive name, e.g. c:\. Windows network paths and other special paths starting with a double separator are not supported (\cloud\drive\path, \?\, etc.). Separators can be \ and / and will be preserved. Newly created separators are always the forward slash /.
On Windows and macOS, the path segment tests are case insensitive.
If the path can not be generated, from path is copied into the to buffer and 0 is returned.
#include <FL/filename.H>  
[.]  
fl_chdir("/var/tmp/somedir"); // set cwd to /var/tmp/somedir  
[.]  
char out[FL_PATH_MAX];
fl_filename_relative(out, sizeof(out), "/var/tmp/somedir/foo.txt"); // out="foo.txt", return=1
fl_filename_relative(out, sizeof(out), "/var/tmp/foo.txt"); // out="../foo.txt", return=1
fl_filename_relative(out, sizeof(out), "foo.txt"); // out="foo.txt", return=0 (no change)
fl_filename_relative(out, sizeof(out), "./.foo.txt"); // out="../foo.txt", return=0 (no change)
fl_filename_relative(out, sizeof(out), "../..foo.txt"); // out="../foo.txt", return=0 (no change)

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>to</th>
<th>resulting relative filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tolen</td>
<td>size of the relative filename buffer</td>
</tr>
<tr>
<td>in</td>
<td>from</td>
<td>absolute filename</td>
</tr>
</tbody>
</table>

Returns

0 if no change, non zero otherwise

See also

fl_filename_relative(char *, to, int tolen, const char *, from, const char *, base)

11.17.3.12 fl_filename_relative() [2/2]

int fl_filename_relative (  
    char * to,  
    int tolen,  
    const char * from,  
    const char * base )

Makes a filename relative to any other directory.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>to</th>
<th>resulting relative filepath</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tolen</td>
<td>size of the relative filename buffer</td>
</tr>
<tr>
<td>in</td>
<td>from</td>
<td>absolute filepath</td>
</tr>
<tr>
<td>in</td>
<td>base</td>
<td>generate filepath relative to this absolute filepath</td>
</tr>
</tbody>
</table>

Returns

0 if no change, non zero otherwise

See also

fl_filename_relative(char *, to, int tolen, const char *, from)

11.17.3.13 fl_filename_setext()

char * fl_filename_setext (  
    char * buf,  
    int buflen,  
    const char * ext )

Replaces the extension in buf of max.

size buflen with the extension in ext.
If there is no '.' in buf, ext is appended.
If ext is NULL, behaves as if it were an empty string ("").

Generated by Doxygen
11.17 File names and URI utility functions

Example

```c
#include <FL/filename.H>
[..]
char buf[FL_PATH_MAX] = "/path/myfile.cxx";
fl_filename_setext(buf, sizeof(buf), ".txt"); // buf[] becomes "/path/myfile.txt"
```

Returns

buf itself for calling convenience.

11.17.3.14 fl_open_uri()

```c
int fl_open_uri (const char ∗uri, char ∗msg, int msglen)
```

Opens the specified Uniform Resource Identifier (URI).

Uses an operating-system dependent program or interface. For URIs using the "ftp", "http", or "https" schemes, the system default web browser is used to open the URI, while "mailto" and "news" URIs are typically opened using the system default mail reader and "file" URIs are opened using the file system navigator.

On success, the (optional) msg buffer is filled with the command that was run to open the URI; on Windows, this will always be "open uri".

On failure, the msg buffer is filled with an English error message.

Note

Platform Specific Issues: Windows
With "file:" based URIs on Windows, you may encounter issues with anchors being ignored. Example: "file://c:/some/index.html#anchor" may open in the browser without the "#anchor" suffix. The behavior seems to vary across different Windows versions. Workaround: open a link to a separate html file that redirects to the desired "file:" URI.

Example

```c
#include <FL/filename.H>
[..]
char errmsg[512];
if ( !fl_open_uri("http://google.com/", errmsg, sizeof(errmsg)) ) {
    char warnmsg[768];
    sprintf(warnmsg, "Error: %s", errmsg);
    fl_alert(warnmsg);
}
```

Parameters

<table>
<thead>
<tr>
<th>uri</th>
<th>The URI to open</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg</td>
<td>Optional buffer which contains the command or error message</td>
</tr>
<tr>
<td>msglen</td>
<td>Length of optional buffer</td>
</tr>
</tbody>
</table>

Returns

1 on success, 0 on failure
Chapter 12

Class Documentation

12.1 Fl_Grid::Cell Class Reference

Public Member Functions

- Fl_Grid_Align align () const
- void align (Fl_Grid_Align align)
- Cell (Fl_Widget ∗w, int row, int col)
- Cell (int row, int col)
- void Cell_()
- short col () const
- short colspan () const
- void colspan (short v)
- void minimum_size (int ∗w, int ∗h) const
- void minimum_size (int w, int h)
- short row () const
- short rowspan () const
- void rowspan (short v)
- Fl_Widget ∗widget () const

Friends

- class Fl_Grid

The documentation for this class was generated from the following file:

- Fl_Grid.H

12.2 Fl_Terminal::CharStyle Class Reference

Public Member Functions

- void attrib (uchar val)
- uchar attrib (void) const
- void bgcolor (Fl_Color val)
- void bgcolor (int r, int g, int b)
- Fl_Color bgcolor (void) const
- void bgcolor_uchar (uchar val)
- void charflags (uchar val)
- uchar charflags (void) const
- CharStyle (bool fontsize_defer)
- int charwidth (void) const
- void clr_charflag (uchar val)
• uchar colorbits_only (uchar inflags) const
• void defaultbgcolor (Fl_Color val)
• Fl_Color defaultbgcolor (void) const
• void defaultfgcolor (Fl_Color val)
• Fl_Color defaultfgcolor (void) const
• void fgcolor (Fl_Color val)
• void fgcolor (int r, int g, int b)
• Fl_Color fgcolor (void) const
• void fgcolor_uchar (uchar val)
• int fontdescent (void) const
• void fontface (Fl_Font val)
• Fl_Font fontface (void) const
• int fontheight (void) const
• void fontsize (Fl_Fontsize val)
• Fl_Fontsize fontsize (void) const
• int onoff (bool flag, Attrib a)
• void set_charflag (uchar val)
• void sgr_blink (bool val)
• void sgr_bold (bool val)
• void sgr_dbld_under (bool val)
• void sgr_dim (bool val)
• void sgr_inverse (bool val)
• void sgr_italic (bool val)
• void sgr_reset (void)
• void sgr_strike (bool val)
• void sgr_underline (bool val)
• void update (void)
• void update_fake (void)

The documentation for this class was generated from the following files:

• Fl_Terminal.H
• Fl_Terminal.cxx

12.3 Fl_GIF_Image::GIF_FRAME::CPAL Struct Reference

Public Attributes

• uchar b
• uchar g
• uchar r

The documentation for this struct was generated from the following file:

• Fl_GIF_Image.H

12.4 Fl_Terminal::Cursor Class Reference

Public Member Functions

• void bgcolor (Fl_Color val)
• Fl_Color bgcolor (void) const
• void col (int val)
• int col (void) const
• int down (void)
• void fgcolor (Fl_Color val)
12.5 Fl_Preferences::Entry Struct Reference

Public Attributes

- char * name
- char * value

The documentation for this struct was generated from the following file:

- Fl_Preferences.H

12.6 Fl_Terminal::EscapeSeq Class Reference

Public Member Functions

- int defvalmax (int dval, int max) const
- void esc_mode (char val)
- char esc_mode (void) const
- bool is_csi (void) const
- int parse (char c)
- bool parse_in_progress (void) const
- void reset (void)
- void restore_cursor (int &row, int &col)
- void save_cursor (int row, int col)
- int total_vals (void) const
- int val (int i) const

Static Public Attributes

- static const int completed = 1
- static const int fail = -1
- static const int maxbuff = 80
- static const int maxvals = 20
- static const int success = 0

The documentation for this class was generated from the following files:

- Fl_Terminal.H
- Fl_Terminal.cxx
12.7  FL Class Reference

The FL is the FLTK global (static) class containing state information and global methods for the current application.

```cpp
#include <Fl.H>
```

### Public Types

- `enum FL_Option {
  OPTION_ARROW_FOCUS = 0,
  OPTION_VISIBLE_FOCUS,
  OPTION_DND_TEXT,
  OPTION_SHOW_TOOLTIPS,
  OPTION_FNFC_USES_GTK,
  OPTION_PRINTER_USES_GTK,
  OPTION_SHOW_SCALING,
  OPTION_FNFC_USES_ZENITY,
  OPTION_LAST
}`

Enumerator for global FLTK options.

### Static Public Member Functions

- `static int abi_check (const int val=FL_ABI_VERSION)`
  Returns whether the runtime library ABI version is correct.
- `static int abi_version ()`
  Returns the compiled-in value of the FL_ABI_VERSION constant.
- `static int add_awake_handler_ (Fl_Awake_Handler, void *)`
  Adds an awake handler for use in `awake()`.
- `static void add_check (Fl_Timeout_Handler, void *=0)`
  FLTK will call this callback just before it flushes the display and waits for events.
- `static void add_clipboard_notify (Fl_Clipboard_Notify_Handler h, void *=0)`
  FLTK will call the registered callback whenever there is a change to the selection buffer or the clipboard.
- `static void add_fd (int fd, Fl_FD_Handler cb, void *=0)`
  Adds file descriptor fd to listen to.
- `static void add_fd (int fd, int when, Fl_FD_Handler cb, void *=0)`
  Adds file descriptor fd to listen to.
- `static void add_handler (Fl_Event_Handler h)`
  Install a function to parse unrecognized events.
- `static void add_idle (Fl_Idle_Handler cb, void *=0)`
  Adds a callback function that is called every time by `Fl::wait()` and also makes it act as though the timeout is zero (this makes `Fl::wait()` return immediately, so if it is in a loop it is called repeatedly, and thus the idle function is called repeatedly).
- `static void add_system_handler (Fl_System_Handler h, void *=data)`
  Install a function to intercept system events.
- `static void add_timeout (double t, Fl_Timeout_Handler cb, void *=data=0)`
  Adds a one-shot timeout callback.
- `static int api_version ()`
  Returns the compiled-in value of the FL_API_VERSION constant.
- `static int arg (int argc, char **argv, int &i)`
  Parse a single switch from `argv`, starting at word `i`.
- `static void args (int argc, char **argv)`
  Parse all command line switches matching standard FLTK options only.
- `static void args (int argc, char **argv, int &i, Fl_Args_Handler cb=0)`
  Parse command line switches using the `cb` argument handler.
- `static int args_to_utf8 (int argc, char **&argv)`
  Convert Windows commandline arguments to UTF-8.
- `static int awake (Fl_Awake_Handler cb, void *=message=0)`
  See `void awake(void* message=0)`.
• static void awake (void *message=0)
  Sends a message pointer to the main thread, causing any pending Fl::wait() call to terminate so that the main thread can retrieve the message and any pending redraws can be processed.

• static void background (uchar, uchar, uchar)
  Changes fl_color(FL_BACKGROUND_COLOR) to the given color, and changes the gray ramp from 32 to 56 to black to white.

• static void background2 (uchar, uchar, uchar)
  Changes the alternative background color.

• static Fl_Widget * belowmouse ()
  Gets the widget that is below the mouse.

• static void belowmouse (Fl_Widget *)
  Sets the widget that is below the mouse.

• static int box_border_radius_max ()
  Get the maximum border radius of all "rounded" boxtypes in pixels.

• static void box_border_radius_max (int R)
  Set the maximum border radius of all "rounded" boxtypes in pixels.

• static Fl_Color box_color (Fl_Color)
  Gets the drawing color to be used for the background of a box.

• static int box_dh (Fl_Boxtype)
  Returns the height offset for the given boxtype.

• static int box_dw (Fl_Boxtype)
  Returns the width offset for the given boxtype.

• static int box_dx (Fl_Boxtype)
  Returns the X offset for the given boxtype.

• static int box_dy (Fl_Boxtype)
  Returns the Y offset for the given boxtype.

• static int box_shadow_width ()
  Get the box shadow width of all "shadow" boxtypes in pixels.

• static void box_shadow_width (int W)
  Set the box shadow width of all "shadow" boxtypes in pixels.

• static bool cairo_autolink_context ()
  Gets the current autolink mode for Cairo support.

• static void cairo_autolink_context (bool alink)
  When FLTK_HAVE_CAIRO is defined and cairo_autolink_context() is true, any current window dc is linked to a current Cairo context.

• static cairo_t * cairo_cc ()
  Gets the current Cairo context linked with a fltk window.

• static void cairo_cc (cairo_t *c, bool own=false)
  Sets the current Cairo context to c.

• static void cairo_flush (cairo_t *c)
  Flush Cairo drawings on Cairo context c.

• static cairo_t * cairo_make_current (Fl_Window *w)
  Provides a Cairo context for window wi.

• static Fl_Callback_Reason callback_reason ()
  Give the reason for calling a callback.

• static int check ()
  Same as Fl::wait(0).

• static void clear_widget_pointer (Fl_Widget const *w)
  Clears a widget pointer in the watch list.

• static int clipboard_contains (const char *type)
  Returns non 0 if the clipboard contains data matching type.
• static int compose (int &del)
      Any text editing widget should call this for each FL_KEYBOARD event.
• static void compose_reset ()
      If the user moves the cursor, be sure to call Fl::compose_reset()!
• static void copy (const char ∗stuff, int len, int destination=0, const char ∗type=Fl::clipboard_plain_text)
      Copies the data pointed to by stuff to the selection buffer (destination is 0), the clipboard (destination is 1), or both (destination is 2).
• static int damage ()
      If true then flush() will do something.
• static void damage (int d)
      If true then flush() will do something.
• static void default_atclose (Fl_Window ∗, void ∗)
      Default callback for window widgets.
• static void delete_widget (Fl_Widget ∗w)
      Schedules a widget for deletion at the next call to the event loop.
• static void disable_im ()
      Disables the system input methods facilities.
• static void display (const char ∗)
      Sets the X or Wayland display to use for all windows.
• static int dnd ()
      Initiate a Drag And Drop operation.
• static int dnd_text_ops ()
      Gets whether drag and drop text operations are supported.
• static void dnd_text_ops (int v)
      Sets whether drag and drop text operations are supported.
• static void do_widget_deletion ()
      Deletes widgets previously scheduled for deletion.
• static int draw_box_active ()
      Determines if the currently drawn box is active or inactive.
• static int draw_GL_text_with_textures ()
      returns whether whether OpenGL uses textures to draw all text.
• static void draw_GL_text_with_textures (int val)
      sets whether OpenGL uses textures to draw all text.
• static void enable_im ()
      Enables the system input methods facilities.
• static int event ()
      Returns the last event that was processed.
• static int event_alt ()
      Returns non-zero if the Alt key is pressed.
• static int event_button ()
      Gets which particular mouse button caused the current event.
• static int event_button1 ()
      Returns non-zero if mouse button 1 is currently held down.
• static int event_button2 ()
      Returns non-zero if button 2 is currently held down.
• static int event_button3 ()
      Returns non-zero if button 3 is currently held down.
• static int event_buttons ()
      Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.
• static int event_clicks ()
      Returns non zero if we had a double click event.
- static void event_clicks (int i)
  Manually sets the number returned by Fl::event_clicks().
- static void * event_clipboard ()
  During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.
- static const char * event_clipboard_type ()
  Returns the type of the pasted data during an FL_PASTE event.
- static int event_command ()
  Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.
- static int event_ctrl ()
  Returns non-zero if the Control key is pressed.
- static Fl_Event_Dispatch event_dispatch ()
  Return the current event dispatch function.
- static void event_dispatch (Fl_Event_Dispatch d)
  Set a new event dispatch function.
- static int event_dx ()
  Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
- static int event_dy ()
  Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
- static int event_inside (const Fl_Widget *)
  Returns whether or not the mouse event is inside a given child widget.
- static int event_inside (int, int, int, int)
  Returns whether or not the mouse event is inside the given rectangle.
- static int event_is_click ()
  Returns non-zero if the mouse has not moved far enough and not enough time has passed since the last FL_PUSH or FL_KEYBOARD event for it to be considered a "drag" rather than a "click".
- static void event_is_click (int i)
  Clears the value returned by Fl::event_is_click().
- static int event_key ()
  Gets which key on the keyboard was last pushed.
- static int event_key (int key)
  Returns true if the given key was held down (or pressed) during the last event.
- static int event_length ()
  Returns the length of the text in Fl::event_text().
- static int event_original_key ()
  Returns the keycode of the last key event, regardless of the NumLock state.
- static int event_shift ()
  Returns non-zero if the Shift key is pressed.
- static int event_state ()
  Returns the keyboard and mouse button states of the last event.
- static int event_state (int mask)
  Returns non-zero if any of the passed event state bits are turned on.
- static const char * event_text ()
  Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.
- static int event_x ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.
- static int event_x_root ()
  Returns the mouse position on the screen of the event.
- static int event_y ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.
- static int event_y_root ()
  Returns the mouse position on the screen of the event.
• static Fl_Window * first_window ()
  Returns the first top-level window in the list of shown() windows.
• static void first_window (Fl_Window *)
  Sets the window that is returned by first_window().
• static void flush ()
  Causes all the windows that need it to be redrawn and graphics forced out through the pipes.
• static Fl_Widget * focus ()
  Gets the current Fl::focus() widget.
• static void focus (Fl_Widget *)
  Sets the widget that will receive FL_KEYBOARD events.
• static void foreground (uchar, uchar, uchar)
  Changes fl_color(FL_FOREGROUND_COLOR).
• static void free_color (Fl_Color i, int overlay=0)
  Frees the specified color from the colormap, if applicable.
• static int get_awake_handler_ (Fl_Awake_Handler &, void *&)
  Gets the last stored awake handler for use in awake().
• static Fl_Box_Draw_F * get_boxtype (Fl_Boxtype)
  Gets the current box drawing function for the specified box type.
• static unsigned get_color (Fl_Color i)
  Returns the RGB value(s) for the given FLTK color index.
• static void get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue)
  Returns the RGB value(s) for the given FLTK color index.
• static void get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue, uchar &alpha)
  Returns the RGBA value(s) for the given FLTK color index.
• static const char * get_font (Fl_Font)
  Gets the string for this face.
• static const char * get_font_name (Fl_Font, int *attributes=0)
  Get a human-readable string describing the family of this face.
• static int get_font_sizes (Fl_Font, int *sizep)
  Return an array of sizes in sizep.
• static int get_key (int key)
  Returns true if the given key is held down now.
• static void get_mouse (int &, int &)
  Return where the mouse is on the screen by doing a round-trip query to the server.
• static void get_system_colors ()
  Read the user preference colors from the system and use them to call Fl::foreground(), Fl::background(), and Fl::background2().
• static int gl_visual (int, int *alist=0)
  This does the same thing as Fl::visual(int) but also requires OpenGL drawing to work.
• static Fl_Window * grab ()
  Returns the window that currently receives all events.
• static void grab (Fl_Window &win)
  See grab(Fl_Window+)
• static void grab (Fl_Window *)
  Selects the window to grab.
• static int h ()
  Returns the height in pixels of the main screen work area.
• static int handle (int, Fl_Window *)
  Handle events from the window system.
• static int handle_ (int, Fl_Window *)
  Handle events from the window system.
• static int **has_check** (Fl_Timeout_Handler, void ∗=0)
  
  Returns 1 if the check exists and has not been called yet, 0 otherwise.

• static int **has_idle** (Fl_Idle_Handler cb, void ∗data=0)

  Returns true if the specified idle callback is currently installed.

• static int **has_timeout** (Fl_Timeout_Handler cb, void ∗data=0)

  Returns true if the timeout exists and has not been called yet.

• static void hide_all_windows ()

  Hide all visible windows to make FLTK leave Fl::run().

• static int is_scheme (const char ∗name)

  Returns whether the current scheme is the given name.

• static void keyboard_screen_scaling (int value)

  Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.

• static int lock ()

  The lock() method blocks the current thread until it can safely access FLTK widgets and data.

• static int menu_linespacing ()

  Gets the default line spacing used by menus.

• static void menu_linespacing (int H)

  Sets the default line spacing used by menus.

• static Fl_Window ∗modal ()

  Returns the top-most modal() window currently shown.

• static Fl_Window ∗next_window (const Fl_Window ∗)

  Returns the next top-level window in the list of shown() windows.

• static Fl_Timestamp now (double offset=0)

  Set a time stamp at this point in time with optional signed offset in seconds.

• static bool option (Fl_Option opt)

  FLTK library options management.

• static void option (Fl_Option opt, bool val)

  Override an option while the application is running.

• static void own_colormap ()

  Makes FLTK use its own colormap.

• static void paste (Fl_Widget &receiver)

  Backward compatibility only.

• static void paste (Fl_Widget &receiver, int source, const char ∗type=Fl::clipboard_plain_text)

  Pastes the data from the selection buffer (source is 0) or the clipboard (source is 1) into receiver.

• static int program_should_quit ()

  Returns non-zero when a request for program termination was received and accepted.

• static void program_should_quit (int should_i)

  Indicate to the FLTK library whether a program termination request was received and accepted.

• static Fl_Widget ∗pushed ()

  Gets the widget that is being pushed.

• static void pushed (Fl_Widget ∗)

  Sets the widget that is being pushed.

• static Fl_Widget ∗readqueue ()

  Reads the default callback queue and returns the first widget.

• static int ready ()

  This is similar to Fl::check() except this does not call Fl::flush() or any callbacks, which is useful if your program is in a state where such callbacks are illegal.

• static void redraw ()

  Redraws all widgets.

• static void release ()

  Releases the current grabbed window, equals grab(0).
• static void release_widget_pointer (Fl_Widget ∗&w)
  Releases a widget pointer from the watch list.
• static int reload_scheme ()
  Called internally when setting a new scheme according to scheme name.
• static void remove_check (Fl_Timeout_Handler, void ∗=0)
  Removes a check callback.
• static void remove_clipboard_notify (Fl_Clipboard_Notify_Handler h)
  Stop calling the specified callback when there are changes to the selection buffer or the clipboard.
• static void remove_fd (int)
  Removes a file descriptor handler.
• static void remove_fd (int, int when)
  Removes a file descriptor handler.
• static void remove_handler (Fl_Event_Handler h)
  Removes a previously added event handler.
• static void remove_idle (Fl_Idle_Handler cb, void ∗data=0)
  Removes the specified idle callback, if it is installed.
• static void remove_system_handler (Fl_System_Handler h)
  Removes a previously added system event handler.
• static void remove_timeout (Fl_Timeout_Handler cb, void ∗data=0)
  Removes a timeout callback from the timer queue.
• static void repeat_timeout (double t, Fl_Timeout_Handler cb, void ∗data=0)
  Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.
• static int run ()
  Calls Fl::wait() repeatedly as long as any windows are displayed.
• static void run_checks ()
• static void run_idle ()
• static const char ∗scheme ()
  See void scheme(const char ∗name)
• static int scheme (const char ∗name)
  Sets the current widget scheme.
• static int screen_count ()
  Gets the total count of available screens.
• static void screen_dpi (float &h, float &v, int n=0)
  Gets the screen resolution in dots-per-inch for the given screen.
• static Fl_Screen_Driver ∗screen_driver ()
  Returns a pointer to the unique Fl_Screen_Driver object of the platform.
• static int screen_num (int x, int y)
  Gets the screen number of a screen that contains the specified screen position x, y.
• static int screen_num (int x, int y, int w, int h)
  Gets the screen number for the screen which intersects the most with the rectangle defined by x, y, w, h.
• static float screen_scale (int n)
  Current value of the GUI scaling factor for screen number n (n [0 , Fl::screen_count()-1])
• static void screen_scale (int n, float factor)
  Sets the value of the GUI scaling factor for screen number n (n [0 , Fl::screen_count()-1]).
• static int screen_scaling_supported ()
  See if scaling factors are supported by this platform.
• static void screen_work_area (int &X, int &Y, int &W, int &H)
  Gets the bounding box of the screen that contains the mouse pointer.
• static void screen_work_area (int &X, int &Y, int &W, int &H, int mx, int my)
  Gets the bounding box of the work area of a screen that contains the specified screen position mx, my.
• static void screen_work_area (int &X, int &Y, int &W, int &H, int n)
**Gets the bounding box of the work area of the given screen.**

- static void `screen_xywh` (int &X, int &Y, int &W, int &H)

  * Gets the bounding box of a screen that contains the mouse pointer.

- static void `screen_xywh` (int &X, int &Y, int &W, int &H, int mx, int my)

  * Gets the bounding box of a screen that contains the specified screen position \(mx, my\).

- static void `screen_xywh` (int &X, int &Y, int &W, int &H, int mx, int my, int mw, int mh)

  * Gets the screen bounding rect for the screen which intersects the most with the rectangle defined by \(mx, my, mw, mh\).

- static void `screen_xywh` (int &X, int &Y, int &W, int &H, int n)

  * Gets the screen bounding rect for the given screen.

- static int scrollbar_size ()

  * Gets the default scrollbar size used by Fl_Browser_, Fl_Help_View, Fl_Scroll, and Fl_Text_Display widgets.

- static void scrollbar_size (int W)

  * Sets the default scrollbar size that is used by the Fl_Browser_, Fl_Help_View, Fl_Scroll, and Fl_Text_Display widgets.

- static double `seconds_between` (Fl_Timestamp &back, Fl_Timestamp &further_back)

  * Return the time in seconds between two time stamps.

- static double `seconds_since` (Fl_Timestamp &then)

  * Return the time in seconds between now and a previously taken time stamp.

- static void `selection` (Fl_Widget &owner, const char *text, int len)

  * Changes the current selection.

- static Fl_Widget * `selection_owner` ()

  * back-compatibility only: Gets the widget owning the current selection

- static void `selection_owner` (Fl_Widget *)

  * Back-compatibility only: The single-argument call can be used to move the selection to another widget or to set the owner to NULL, without changing the actual text of the selection.

- static int `selection_to_clipboard` ()

  * Returns the current selection to clipboard mode.

- static void `selection_to_clipboard` (int mode)

  * Copies selections on X11 directly to the clipboard if enabled.

- static void `set_abort` (Fl_Abort_Handler f)

  * For back compatibility, sets the void Fl::fatal handler callback.

- static void `set_atclose` (Fl_Atclose_Handler f)

  * For back compatibility, sets the Fl::atclose handler callback.

- static void `set_box_color` (Fl_Color)

  * Sets the drawing color for the box that is currently drawn.

- static void `set_boxtype` (Fl_Boxtype, Fl_Box_Draw_F *, uchar, uchar, uchar, uchar)

  * Copies the from boxtype.

- static void `set_boxtype` (Fl_Boxtype, Fl_Boxtype from)

  * Sets the function to call to draw a specific boxtype.

- static void `set_color` (Fl_Color, unsigned c)

  * Sets an entry in the fl_color index table.

- static void `set_color` (Fl_Color, uchar, uchar, uchar)

  * Sets an entry in the fl_color index table.

- static void `set_color` (Fl_Color, uchar, uchar, uchar, uchar)

  * Sets an entry in the fl_color index table.

- static void `set_font` (Fl_Font, const char *)

  * Changes a face.

- static void `set_font` (Fl_Font, Fl_Font)

  * Copies one face to another.

- static Fl_Font `set_fonts` (const char * = 0)

  * FLTK will open the display, and add every fonts on the server to the face table.

- static void `set_idle` (Fl_Old_Idle_Handler cb)
Sets an idle callback.

- **static void set_labeltype (Fl_Labeltype, Fl_Label_Draw_F *, Fl_Label_Measure_F *)**
  Sets the functions to call to draw and measure a specific labeltype.

- **static void set_labeltype (Fl_Labeltype, Fl_Labeltype from)**
  Sets the functions to call to draw and measure a specific labeltype.

- **static int system (const char *command)**
  Run a command line on the computer.

- **static Fl_System_Driver *system_driver ()**
  Returns a pointer to the unique Fl_System_Driver object of the platform.

- **static int test_shortcut (Fl_Shortcut)**
  Tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value (described in Fl_Button).

- **static void *thread_message ()**
  The thread_message() method returns the last message that was sent from a child by the awake() method.

- **static long ticks_between (Fl_Timestamp &back, Fl_Timestamp &further_back)**
  Return the time in ticks (60Hz) between two time stamps.

- **static long ticks_since (Fl_Timestamp &then)**
  Return the time in ticks (60Hz) between now and a previously taken time stamp.

- **static void unlock ()**
  The unlock() method releases the lock that was set using the lock() method.

- **static int use_high_res_GL ()**
  returns whether GL windows should be drawn at high resolution on Apple computers with retina displays.

- **static void use_high_res_GL (int val)**
  sets whether GL windows should be drawn at high resolution on Apple computers with retina displays.

- **static double version ()**
  Returns the compiled-in value of the FL_VERSION constant.

- **static int visible_focus ()**
  Gets or sets the visible keyboard focus on buttons and other non-text widgets.

- **static void visible_focus (int v)**
  Gets or sets the visible keyboard focus on buttons and other non-text widgets.

- **static int visual (int)**
  Selects a visual so that your graphics are drawn correctly.

- **static int w ()**
  Returns the width in pixels of the main screen work area.

- **static int wait ()**
  Waits until "something happens" and then returns.

- **static double wait (double time)**
  Waits a maximum of time_to_wait seconds or until "something happens".

- **static void watch_widget_pointer (Fl_Widget *&w)**
  Adds a widget pointer to the widget watch list.

- **static int x ()**
  Returns the leftmost x coordinate of the main screen work area.

- **static int y ()**
  Returns the topmost y coordinate of the main screen work area.
Static Public Attributes

- static void(* atclose)(Fl_Window *, void *)
  
  Back compatibility: default window callback handler.

- static char const * const clipboard_image = "image"
  
  Denotes image data.

- static char const * const clipboard_plain_text = "text/plain"
  
  Denotes plain textual data.

- static void(* error)(const char *, ...) = Fl_System_Driver::error
  
  FLTK calls Fl::error() to output a normal error message.

- static void(* fatal)(const char *, ...) = Fl_System_Driver::fatal
  
  FLTK calls Fl::fatal() to output a fatal error message.

- static const char * const help = helpmsg+13
  
  Usage string displayed if Fl::args() detects an invalid argument.

- static void(* idle)()
  
  The currently executing idle callback function: DO NOT USE THIS DIRECTLY!

- static void(* warning)(const char *, ...) = Fl_System_Driver::warning
  
  FLTK calls Fl::warning() to output a warning message.

Friends

- class Fl_System_Driver

12.7.1 Detailed Description

The Fl is the FLTK global (static) class containing state information and global methods for the current application.

12.7.2 Member Enumeration Documentation

12.7.2.1 Fl_Option

enum Fl::Fl_Option

Enumerator for global FLTK options.

These options can be set system wide, per user, or for the running application only.

See also

- Fl::option(Fl_Option, bool)
- Fl::option(Fl_Option)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTION_ARROW_FOCUS</td>
<td>When switched on, moving the text cursor beyond the start or end of a text</td>
</tr>
<tr>
<td></td>
<td>in a text widget will change focus to the next text widget. (This is</td>
</tr>
<tr>
<td></td>
<td>considered ‘old’ behavior)</td>
</tr>
<tr>
<td></td>
<td>When switched off (default), the cursor will stop at the end of the text.</td>
</tr>
<tr>
<td></td>
<td>Pressing Tab or Ctrl-Tab will advance the keyboard focus.</td>
</tr>
<tr>
<td></td>
<td>See also: Fl::Input_::tab_nav()</td>
</tr>
<tr>
<td>OPTION_VISIBLE_FOCUS</td>
<td>If visible focus is switched on (default), FLTK will draw a dotted rectangle</td>
</tr>
<tr>
<td></td>
<td>inside the widget that will receive the next keystroke. If switched off, no</td>
</tr>
<tr>
<td></td>
<td>indicator will be drawn and keyboard navigation is disabled.</td>
</tr>
<tr>
<td>OPTION_DND_TEXT</td>
<td>If text drag-and-drop is enabled (default), the user can select and drag text</td>
</tr>
<tr>
<td></td>
<td>from any text widget. If disabled, no dragging is possible, however dropping</td>
</tr>
<tr>
<td></td>
<td>text from other applications still works.</td>
</tr>
<tr>
<td>OPTION_SHOW_TOOLTIP</td>
<td>If tooltips are enabled (default), hovering the mouse over a widget with a</td>
</tr>
<tr>
<td></td>
<td>tooltip text will open a little tooltip window until the mouse leaves the</td>
</tr>
<tr>
<td></td>
<td>widget. If disabled, no tooltip is shown.</td>
</tr>
</tbody>
</table>

Generated by Doxygen
**Enumerator**

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTION_FNFC_USES_GTK</td>
<td>When switched on (default), Fl_Native_File_Chooser runs GTK file dialogs if the GTK library is available on the platform (linux/unix only). When switched off, GTK file dialogs aren't used even if the GTK library is available.</td>
</tr>
<tr>
<td>OPTION_PRINTER_USES_GTK</td>
<td>When switched on (default), Fl_Printer runs the GTK printer dialog if the GTK library is available on the platform (linux/unix only). When switched off, the GTK printer dialog isn't used even if the GTK library is available.</td>
</tr>
<tr>
<td>OPTION_SHOW_SCALING</td>
<td>When switched on (default), the library shows in a transient yellow window the zoom factor value. When switched off, no such window gets displayed.</td>
</tr>
<tr>
<td>OPTION_FNFC_USES_ZENITY</td>
<td>Meaningful for the Wayland/X11 platform only. When switched on (default), the library uses a Zenity-based file dialog. When switched off, the GTK file dialog is used instead.</td>
</tr>
<tr>
<td>OPTION_LAST</td>
<td>For internal use only.</td>
</tr>
</tbody>
</table>

### 12.7.3 Member Function Documentation

#### 12.7.3.1 abi_check()

```c
static int Fl::abi_check (
         const int val = FL_ABI_VERSION ) [inline], [static]
```

Returns whether the runtime library ABI version is correct. This enables you to check the ABI version of the linked FLTK library at runtime. Returns 1 (true) if the compiled ABI version (in the header files) and the linked library ABI version (used at runtime) are the same, 0 (false) otherwise. Argument `val` can be used to query a particular library ABI version. Use for instance 10303 to query if the runtime library is compatible with FLTK ABI version 1.3.3. This is rarely useful. The default `val` argument is FL_ABI_VERSION, which checks the version defined at configure time (i.e. in the header files at program compilation time) against the linked library version used at runtime. This is particularly useful if you linked with a shared object library, but it also concerns static linking.

See also

- Fl::abi_version()

#### 12.7.3.2 abi_version()

```c
int Fl::abi_version ( ) [static]
```

Returns the compiled-in value of the FL_ABI_VERSION constant. This is useful for checking the version of a shared library.

#### 12.7.3.3 add_check()

```c
void Fl::add_check ( Fl_Timeout_Handler cb,
         void * argp = 0 ) [static]
```

FLTK will call this callback just before it flushes the display and waits for events. This is different than an idle callback because it is only called once, then FLTK calls the system and tells it not to return until an event happens. This can be used by code that wants to monitor the application's state, such as to keep a display up to date. The advantage of using a check callback is that it is called only when no events are pending. If events are coming in quickly, whole blocks of them will be processed before this is called once. This can save significant time and avoid the application falling behind the events.

Sample code:

```c
bool state_changed; // anything that changes the display turns this on
void callback(void*) {
```
if (!state_changed) return;
state_changed = false;
do_expensive_calculation();
widget->redraw();
}

main() {
Fl::add_check{callback};
return Fl::run();
}

12.7.3.4 add_fd() [1/2]
void Fl::add_fd (  
int fd,
Fl_FD_Handler cb,
void * d = 0 ) [static]

Adds file descriptor fd to listen to.
See Fl::add_fd(int fd, int when, Fl_FD_Handler cb, void * = 0) for details

12.7.3.5 add_fd() [2/2]
void Fl::add_fd (  
int fd,
int when,
Fl_FD_Handler cb,
void * d = 0 ) [static]

Adds file descriptor fd to listen to.
When the fd becomes ready for reading Fl::wait() will call the callback and then return. The callback is passed the
fd and the arbitrary void* argument.
This version takes a when bitfield, with the bits FL_READ, FL_WRITE, and FL_EXCEPT defined, to indicate when
the callback should be done.
There can only be one callback of each type for a file descriptor. Fl::remove_fd() gets rid of all the callbacks for a
given file descriptor.
Under UNIX/Linux/macOS any file descriptor can be monitored (files, devices, pipes, sockets, etc.). Due to limita-
tions in Microsoft Windows, Windows applications can only monitor sockets.
Under macOS, Fl::add_fd() opens the display if that’s not been done before.

12.7.3.6 add_idle()
void Fl::add_idle (  
Fl_Idle_Handler cb,
void * data = 0 ) [static]

Adds a callback function that is called every time by Fl::wait() and also makes it act as though the timeout is zero
(this makes Fl::wait) return immediately, so if it is in a loop it is called repeatedly, and thus the idle function is called
repeatedly).
The idle function can be used to get background processing done.
You can have multiple idle callbacks. To remove an idle callback use Fl::remove_idle().
Fl::wait() and Fl::check() call idle callbacks, but Fl::ready() does not.
The idle callback can call any FLTK functions, including Fl::wait(), Fl::check(), and Fl::ready().
FLTK will not recursively call the idle callback.

12.7.3.7 add_timeout()
void Fl::add_timeout (  
double time,
Fl_Timeout_Handler cb,
void * data = 0 ) [static]

Adds a one-shot timeout callback.
The callback function cb will be called by Fl::wait() at time seconds after this function is called. The callback
function must have the signature Fl_Timeout_Handler. The optional data argument is passed to the callback
(default: NULL).
The timer is removed from the timer queue before the callback function is called. It is safe to reschedule the timeout inside the callback function.

You can have multiple timeout callbacks, even the same timeout callback with different timeout values and/or different data values. They are all considered different timer objects.

To remove a timeout while it is active (pending) use Fl::remove_timeout().

If you need more accurate, repeated timeouts, use Fl::repeat_timeout() to reschedule the subsequent timeouts. Please see Fl::repeat_timeout() for an example.

Since version 1.4, a timeout can be started from a child thread under the condition that the call to Fl::add_timeout is wrapped in Fl::lock() and Fl::unlock().

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>delta time in seconds until the timer expires</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cb</td>
<td>callback function</td>
</tr>
<tr>
<td>in</td>
<td>data</td>
<td>optional user data (default: NULL)</td>
</tr>
</tbody>
</table>

See also

Fl_Timeout_Handler
Fl::repeat_timeout(double time, Fl_Timeout_Handler cb, void *data)
Fl::remove_timeout(Fl_Timeout_Handler cb, void *data)
Fl::has_timeout(Fl_Timeout_Handler cb, void *data)

12.7.3.8 api_version()

int Fl::api_version ( ) [static]

Returns the compiled-in value of the FL_API_VERSION constant.

This is useful for checking the version of a shared library.

12.7.3.9 arg()

int Fl::arg (int argc, char **argv, int &i) [static]

Parse a single switch from argv, starting at word i.

Returns the number of words eaten (1 or 2, or 0 if it is not recognized) and adds the same value to i.

This is the default argument handler used internally by Fl::args(...), but you can use this function if you prefer to step through the standard FLTK switches yourself.

All standard FLTK switches except -bg2 may be abbreviated to just one letter and case is ignored:

- -bg color or -background color
  Sets the background color using Fl::background().

- -bg2 color or -background2 color
  Sets the secondary background color using Fl::background2().

- -display host:n.n
  Sets the X display to use; this option is silently ignored under Windows and MacOS.

- -dnd and -nodnd
  Enables or disables drag and drop text operations using Fl::dnd_text_ops().

- -fg color or -foreground color
  Sets the foreground color using Fl::foreground().

- -geometry WxH+X+Y
  Sets the initial window position and size according to the standard X geometry string.
• -iconic
  Iconifies the window using Fl_Window::iconize().

• -kbd and -nokbd
  Enables or disables visible keyboard focus for non-text widgets using Fl::visible_focus().

• -name string
  Sets the window class using Fl_Window::xclass().

• -scheme string
  Sets the widget scheme using Fl::scheme().

• -title string
  Sets the window title using Fl_Window::label().

• -tooltips and -notooltips
  Enables or disables tooltips using Fl_Tooltip::enable().

Color values are commonly given as three digit or six digit hex numbers.

• The order of fg, bg, and bg2 in the command line does not matter

• There is no way at the moment to set the selection color.

• Setting the bg2 color also changes the fg color to have sufficient contrast

• Explicitly setting fg color overrides the bg2/contrast constraint.

• Setting the bg color will update the color lookup table for the gray ramp, so color index values can stay the same for all apps, it's just mapped to different RGB values.

• The calculation of the gray ramp is only based on the bg color, so there is no way at the moment to create an inverted (dark mode) ramp.

• Consequently, setting bg to black creates a an all-black ramp, setting a somewhat dark bg color creates a extremely dark ramp.

• Setting the bg has no influence on bg2 or fg.

If your program requires other switches in addition to the standard FLTK options, you will need to pass your own argument handler to Fl::args(int,char**,int&,Fl_Args_Handler) explicitly.

See also
fl_parse_color(const char* p, uchar& r, uchar& g, uchar& b) to see how color values can be defined

12.7.3.10  args() [1/2]

void Fl::args {
  int argc,
  char ** argv ) [static]
Parse all command line switches matching standard FLTK options only. It parses all the switches, and if any are not recognized it calls Fl::abort(Fl::help), i.e. unlike the long form, an unrecognized switch generates an error message and causes the program to exit.

12.7.3.11  args() [2/2]

int Fl::args {
  int argc,
  char ** argv,
  int & i,
  Fl_Args_Handler cb = 0 ) [static]
Parse command line switches using the cb argument handler.
Returns 0 on error, or the number of words processed.

FLTK provides this as an entirely optional command line switch parser. You don't have to call it if you don't want to. Everything it can do can be done with other calls to FLTK.

To use the switch parser, call Fl::args(...) near the start of your program. This does not open the display, instead switches that need the display open are stashed into static variables. Then you must display your first window by calling window->show(argc, argv), which will do anything stored in the static variables.

Providing an argument handler callback cb lets you define your own switches. It is called with the same argc and argv, and with i set to the index of the switch to be processed. The cb handler should return zero if the switch is unrecognized, and not change i. It should return non-zero to indicate the number of words processed if the switch is recognized, i.e. 1 for just the switch, and more than 1 for the switch plus associated parameters. i should be incremented by the same amount.

The cb handler is called before any other tests, so you can also override any standard FLTK switch (this is why FLTK can use very short switches instead of the long ones all other toolkits force you to use). See Fl::arg() for descriptions of the standard switches.

On return i is set to the index of the first non-switch. This is either:

- The first word that does not start with '-'.
- The word '-' (used by many programs to name stdin as a file)
- The first unrecognized switch (return value is 0).
- argc

The return value is i unless an unrecognized switch is found, in which case it is zero. If your program takes no arguments other than switches you should produce an error if the return value is less than argc.

A usage string is displayed if Fl::args() detects an invalid argument on the command-line. You can change the message by setting the Fl::help pointer.

A very simple command line parser can be found in examples/howto-parse-args.cxx

The simpler Fl::args(int argc, char **argv) form is useful if your program does not have command line switches of its own.

### 12.7.3.12 args_to_utf8()

```cpp
int Fl::args_to_utf8 ( int argc, char ***argv ) [static]
```

Convert Windows commandline arguments to UTF-8.

**Note**

This function does nothing on other (non-Windows) platforms, hence you may call it on all platforms or only on Windows by using platform specific code like `#ifdef _WIN32` etc. - it's your choice. Calling it on other platforms returns quickly w/o wasting much CPU time.

This function must be called on Windows platforms in main() before the array argv is used if your program uses any commandline argument strings (these should be UTF-8 encoded). This applies also to standard FLTK commandline arguments like "-name" (class name) and "-title" (window title in the title bar).

Unfortunately Windows neither provides commandline arguments in UTF-8 encoding nor as Windows "Wide Character" strings in the standard main() and/or the Windows specific WinMain() function.

On Windows platforms (no matter which build system) this function calls a Windows specific function to retrieve commandline arguments as Windows "Wide Character" strings, converts these strings to an internally allocated buffer (or multiple buffers) and returns the result in argv. For implementation details please refer to the source code; however these details may be changed in the future.

Note that argv is provided by reference so it can be overwritten.

In the recommended simple form the function overwrites the variable argv and allocates a new array of strings pointed to by argv. You may use this form on all platforms and it is as simple as adding one line to old programs to make them work with international (UTF-8) commandline arguments.

```cpp
int main(int argc, char **argv) {  
  Fl::args_to_utf8(argc, argv);  // add this line
  // ... use argc and argv, e.g. for commandline parsing
  window->show(argc, argv);
  return Fl::run();
}
```
For an example see `examples/howto-parse-args.cxx` in the FLTK sources.

If you want to retain the original `argc` and `argv` variables the following slightly longer and more complicated code works as well on all platforms.

```c
int main(int argc, char **argv)
{
    char **argvn = argv; // must copy argv to work on all platforms
    int argcn = Fl::args_to_utf8(argc, argvn);
    // ... use argcn and argvn, e.g. for commandline parsing
    window->show(argcn, argvn);
    return Fl::run();
}
```

**Parameters**

| in | argc | used only on non-Windows platforms |
| out | argv | modified only on Windows platforms |

**Returns**

argument count (always the same as argc)

Since

1.4.0

### 12.7.3.13 `background()`

```c
void Fl::background (uchar r, uchar g, uchar b) [static]
```

Changes `fl_color(FL_BACKGROUND_COLOR)` to the given color, and changes the gray ramp from 32 to 56 to black to white.

These are the colors used as backgrounds by almost all widgets and used to draw the edges of all the boxtypes.

### 12.7.3.14 `background2()`

```c
void Fl::background2 (uchar r, uchar g, uchar b) [static]
```

Changes the alternative background color.

This color is used as a background by `Fl_Input` and other text widgets.

This call may change `fl_color(FL_FOREGROUND_COLOR)` if it does not provide sufficient contrast to `FL_←BACKGROUND2_COLOR`.

### 12.7.3.15 `box_border_radius_max()` [1/2]

```c
static int Fl::box_border_radius_max ( ) [inline], [static]
```

Get the maximum border radius of all "rounded" boxtypes in pixels.

Since

1.4.0

### 12.7.3.16 `box_border_radius_max()` [2/2]

```c
static void Fl::box_border_radius_max (int R) [inline], [static]
```

Set the maximum border radius of all "rounded" boxtypes in pixels.

Must be at least 5, default = 15.
Note

This does **not** apply to the "round" boxtypes which have really round sides (i.e. composed of half circles) as opposed to "rounded" boxtypes that have only rounded corners with a straight border between corners.

The box border radius of "rounded" boxtypes is typically calculated as about 2/5 of the box height or width, whichever is smaller. The upper limit can be set by this method for all "rounded" boxtypes.

Since

1.4.0

### 12.7.3.17 box_color()

```cpp
Fl_Color Fl::box_color ( Fl_Color c ) [static]
```

Gets the drawing color to be used for the background of a box.

This method is only useful inside box drawing code. It returns the color to be used, either `fl_inactive(c)` if the widget is `!active_r()` or `c` otherwise.

### 12.7.3.18 box_dh()

```cpp
int Fl::box_dh ( Fl_Boxtype t ) [static]
```

Returns the height offset for the given boxtype.

See also

`box_dy()`.

### 12.7.3.19 box_dw()

```cpp
int Fl::box_dw ( Fl_Boxtype t ) [static]
```

Returns the width offset for the given boxtype.

See also

`box_dy()`.

### 12.7.3.20 box_dx()

```cpp
int Fl::box_dx ( Fl_Boxtype t ) [static]
```

Returns the X offset for the given boxtype.

See also

`box_dy()`.

### 12.7.3.21 box_dy()

```cpp
int Fl::box_dy ( Fl_Boxtype t ) [static]
```

Returns the Y offset for the given boxtype.

These functions return the offset values necessary for a given boxtype, useful for computing the area inside a box's borders, to prevent over drawing the borders.

For instance, in the case of a boxtype like `FL_DOWN_BOX` where the border width might be 2 pixels all around, the above functions would return 2, 2, 4, and 4 for `box_dx, box_dy, box_dw, and box_dh` respectively.

An example to compute the area inside a widget's box:

```cpp
int X = yourwidget->x() + Fl::box_dx(yourwidget->box());
```
int Y = yourwidget->y() + Fl::box_dy(yourwidget->box());
int W = yourwidget->w() - Fl::box_dw(yourwidget->box());
int H = yourwidget->h() - Fl::box_dh(yourwidget->box());

These functions are mainly useful in the draw() code for deriving custom widgets, where one wants to avoid drawing
over the widget's own border box().

12.7.3.22 box_shadow_width() [1/2]
static int Fl::box_shadow_width ( ) [inline], [static]
Get the box shadow width of all "shadow" boxtypes in pixels.
Since

1.4.0

12.7.3.23 box_shadow_width() [2/2]
static void Fl::box_shadow_width ( 
    int W ) [inline], [static]
Set the box shadow width of all "shadow" boxtypes in pixels.
Must be at least 1, default = 3. There is no upper limit.
Since

1.4.0

12.7.3.24 check()
int Fl::check ( ) [static]
Same as Fl::wait(0).
Calling this during a big calculation will keep the screen up to date and the interface responsive:
while (!calculation_done()) {
    calculate();
    Fl::check();
    if (user_hit_abort_button()) break;
}
This returns non-zero if any windows are displayed, and 0 if no windows are displayed (this is likely to change in
future versions of FLTK).

12.7.3.25 display()
void Fl::display ( 
    const char * d ) [static]
Sets the X or Wayland display to use for all windows.
This sets the environment variable $DISPLAY or $WAYLAND_DISPLAY to the passed string, so this only works
before you show() the first window or otherwise open the display.
This does nothing on other platforms.

12.7.3.26 dnd_text_ops() [1/2]
static int Fl::dnd_text_ops ( ) [inline], [static]
Gets whether drag and drop text operations are supported.
This returns whether selected text can be dragged from text fields or dragged within a text field as a cut/paste
shortcut.

12.7.3.27 dnd_text_ops() [2/2]
static void Fl::dnd_text_ops ( 
    int v ) [inline], [static]
Sets whether drag and drop text operations are supported.
This specifically affects whether selected text can be dragged from text fields or dragged within a text field as a
cut/paste shortcut.
12.7.3.28 draw_box_active()

int Fl::draw_box_active ( ) [static]
Determines if the currently drawn box is active or inactive.
If inactive, the box color should be changed to the inactive color.

See also
   Fl::box_color(Fl_Color c)

12.7.3.29 draw_GL_text_with_textures() [1/2]

static int Fl::draw_GL_text_with_textures ( ) [inline], [static]
returns whether OpenGL uses textures to draw all text.
Default is yes.

See also
   draw_GL_text_with_textures(int val)

Version
   1.4.0

12.7.3.30 draw_GL_text_with_textures() [2/2]

static void Fl::draw_GL_text_with_textures ( int val ) [inline], [static]
sets whether OpenGL uses textures to draw all text.
By default, FLTK draws OpenGL text using textures, if the necessary hardware support is available. Call Fl::
draw_GL_text_with_textures(0) once in your program before the first call to gl_font() to have FLTK
draw instead OpenGL text using a legacy, platform-dependent procedure. It's recommended not to deactivate
textures under the MacOS platform because the MacOS legacy procedure is extremely rudimentary.

Parameters

| val | use 0 to prevent FLTK from drawing GL text with textures |

See also
   gl_texture_pile_height(int max)

Version
   1.4.0

12.7.3.31 flush()

void Fl::flush ( ) [static]
Causes all the windows that need it to be redrawn and graphics forced out through the pipes.
This is what wait() does before looking for events.
Note: in multi-threaded applications you should only call Fl::flush() from the main thread. If a child thread needs to
triggleredraw event, it should instead call Fl::awake() to get the main thread to process the event queue.

12.7.3.32 get_system_colors()

void Fl::get_system_colors ( ) [static]
Read the user preference colors from the system and use them to call Fl::foreground(), Fl::background(), and
Fl::background2().
This is done by Fl_Window::show(argc,argv) before applying the -fg and -bg switches. On X this reads some common values from the Xdefaults database. KDE users can set these values by running the "krdb" program, and newer versions of KDE set this automatically if you check the "apply style to other X programs" switch in their control panel.

### 12.7.3.33 gl_visual()

```cpp
static int Fl::gl_visual (int, int∗ alist = 0) [static]
```

This does the same thing as Fl::visual(int) but also requires OpenGL drawing to work. This must be done if you want to draw in normal windows with OpenGL with gl_start() and gl_end(). It may be useful to call this so your X windows use the same visual as an Fl_Gl_Window, which on some servers will reduce colormap flashing.

See Fl_Gl_Window for a list of additional values for the argument.

### 12.7.3.34 has_timeout()

```cpp
int Fl::has_timeout (Fl_Timeout_Handler cb, void∗ data = 0) [static]
```

Returns true if the timeout exists and has not been called yet.
Both arguments cb and data must match with at least one timer in the queue of active timers to return true (1).

**Note**

It is a known inconsistency that Fl::has_timeout() does not use the data argument as a wildcard (match all) if it is zero (NULL) which Fl::remove_timeout() does. This is so for backwards compatibility with FLTK 1.3.x. Therefore using 0 (zero, NULL) as the timeout data value is discouraged unless you're sure that you don't need to use Fl::has_timeout(callback, (void ∗)0); or Fl::remove_timeout(callback, (void ∗)0);.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cb</th>
<th>Timer callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>User data</td>
</tr>
</tbody>
</table>

**Returns**

whether the timer was found in the queue

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>not found</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>found</td>
</tr>
</tbody>
</table>

### 12.7.3.35 hide_all_windows()

```cpp
void Fl::hide_all_windows ( ) [static]
```

Hide all visible windows to make FLTK leave Fl::run(). Fl::run() will run as long as there are visible windows. Call Fl::hide_all_windows() to hide (close) all currently shown (visible) windows, effectively terminating the Fl::run() loop.

See also

Fl::run()
Since

1.4.0

**12.7.3.36 is_scheme()**

static int Fl::is_scheme ( const char * name ) [inline], [static]

Returns whether the current scheme is the given name.
This is a fast inline convenience function to support scheme-specific code in widgets, e.g. in their draw() methods, if required.
Use a valid scheme name, not NULL (although NULL is allowed, this is not a useful argument - see below).
If Fl::scheme() has not been set or has been set to the default scheme ("none" or "base"), then this will always return 0 regardless of the argument, because Fl::scheme() is NULL in this case.

Note

The stored scheme name is always lowercase, and this method will do a case-sensitive compare, so you must provide a lowercase string to return the correct value. This is intentional for performance reasons.

Example:

```c
if (Fl::is_scheme("gtk+")) { your_code_here(); }
```

Parameters

| in name | lowercase string of requested scheme name. |

Returns

1 if the given scheme is active, 0 otherwise.

See also

Fl::scheme(const char *name)

**12.7.3.37 menu_linespacing() [1/2]**

int Fl::menu_linespacing ( ) [static]

Gets the default line spacing used by menus.

Returns

The default line spacing, in pixels.

**12.7.3.38 menu_linespacing() [2/2]**

void Fl::menu_linespacing ( int H ) [static]

Sets the default line spacing used by menus.
Default is 4.

Parameters

| in H | The new default line spacing between menu items, in pixels. |

**12.7.3.39 now()**

Fl_Timestamp Fl::now (  

Generated by Doxygen
Set a time stamp at this point in time with optional signed offset in seconds.
The time stamp is an opaque type and does not represent the time of day or some time and date in the calendar. It is used with `Fl::seconds_between()` and `Fl::seconds_since()` to measure elapsed time.
```cpp
Fl_Timestamp start = Fl::now();
// do something
double s = Fl::seconds_since(start);
printf("That operation took %g seconds\n", s);
```
Depending on the system the resolution may be milliseconds or microseconds. Under certain conditions (particularly on Windows) the value in member `sec` may wrap around and does not represent a real time (maybe runtime of the system). Function `seconds_since()` below uses this to subtract two timestamps which is always a correct delta time with milliseconds or microseconds resolution.

**Parameters**

| offset | optional signed offset in seconds added to the current time |

**Returns**

this moment in time offset by `offset` as an opaque time stamp

**Todo**

`Fl::system_driver()->gettime()` was implemented for the Forms library and has a limited resolution (on Windows: milliseconds). On POSIX platforms it uses `gettimeofday()` with microsecond resolution. A new function could use a better resolution on Windows with its multimedia timers which requires a new dependency: `winmm.lib` (dll). This could be a future improvement, maybe set as a build option or generally (requires Win95 or 98?).

**See also**

- `Fl::seconds_since(Fl_Timestamp& then)`
- `Fl::seconds_between(Fl_Timestamp& back, Fl_Timestamp& further_back)`
- `Fl::ticks_since(Fl_Timestamp& then)`
- `Fl::ticks_between(Fl_Timestamp& back, Fl_Timestamp& further_back)`

### 12.7.3.40 option() [1/2]

```cpp
bool Fl::option (Fl_Option opt) [static]
```

FTLK library options management.

This function needs to be documented in more detail. It can be used for more optional settings, such as using a native file chooser instead of the FLTK one wherever possible, disabling tooltips, disabling visible focus, disabling FLTK file chooser preview, etc.

There should be a command line option interface.

There should be an application that manages options system wide, per user, and per application.

**Example:**

```cpp
if ( Fl::option(Fl::OPTION_ARROW_FOCUS) )
  { ..on.. }
else
  { ..off.. }
```

**Note**

As of FLTK 1.3.0, options can be managed within fluid, using the menu `Edit/Global FLTK Settings`.

**Parameters**

| opt | which option |

Generated by Doxygen
Returns
true or false

See also
enum Fl::Fl_Option
Fl::option(Fl_Option, bool)

Since
FLTK 1.3.0

12.7.3.41 option() [2/2]

void Fl::option (  
    Fl_Option opt,  
    bool val ) [static]

Override an option while the application is running.
This function does not change any system or user settings.
Example:
Fl::option(Fl::OPTION_ARROW_FOCUS, true); // on
Fl::option(Fl::OPTION_ARROW_FOCUS, false); // off

Parameters

<table>
<thead>
<tr>
<th>opt</th>
<th>which option</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>set to true or false</td>
</tr>
</tbody>
</table>

See also
enum Fl::Fl_Option
bool Fl::option(Fl_Option)

12.7.3.42 own_colormap()

void Fl::own_colormap ( ) [static]
Makes FLTK use its own colormap.
This may make FLTK display better and will reduce conflicts with other programs that want lots of colors. However
the colors may flash as you move the cursor between windows.
This does nothing if the current visual is not colormapped.

12.7.3.43 program_should_quit() [1/2]

static int Fl::program_should_quit ( ) [inline], [static]
Returns non-zero when a request for program termination was received and accepted.
On the MacOS platform, the "Quit xxx" item of the application menu is such a request, that is considered ac-
cepted when all windows are closed. On other platforms, this function returns 0 until Fl::program_should←
quit(1) is called.

Version
1.4.0

12.7.3.44 program_should_quit() [2/2]

static void Fl::program_should_quit (  
    int should_i ) [inline], [static]
Indicate to the FLTK library whether a program termination request was received and accepted. A program may set this to 1, for example, while performing a platform-independent command asking the program to cleanly terminate, similarly to the "Quit xxx" item of the application menu under MacOS.

Version
1.4.0

12.7.3.45  readqueue()

Fl_Widget * Fl::readqueue ( ) [static]
Reads the default callback queue and returns the first widget.
All Fl_Widgets that don't have a callback defined use the default callback static Fl_Widget::default_callback() that puts a pointer to the widget in a queue. This method reads the oldest widget out of this queue.
The queue (FIFO) is limited (currently 20 items). If the queue overflows, the oldest entry (Fl_Widget *) is discarded. Relying on the default callback and reading the callback queue with Fl::readqueue() is not recommended. If you need a callback, you should set one with Fl_Widget::callback(Fl_Callback *, void *) or one of its variants.

See also
   Fl_Widget::callback()
   Fl_Widget::callback(Fl_Callback *, void *)
   Fl_Widget::default_callback()

12.7.3.46  ready()

int Fl::ready ( ) [static]
This is similar to Fl::check() except this does not call Fl::flush() or any callbacks, which is useful if your program is in a state where such callbacks are illegal.
This returns true if Fl::check() would do anything (it will continue to return true until you call Fl::check() or Fl::wait()).

while (!calculation_done()) {
   calculate();
   if (Fl::ready()) {
      do_expensive_cleanup();
      Fl::check();
      if (user_hit_abort_button()) break;
   }
}

12.7.3.47  release()

static void Fl::release ( ) [inline], [static]
Releases the current grabbed window, equals grab(0).

Deprecated  Use Fl::grab(0) instead.

See also
   grab(Fl_Window *)

12.7.3.48  reload_scheme()

int Fl::reload_scheme ( ) [static]
Called internally when setting a new scheme according to scheme name.
Loads or reloads the current scheme selection.

Returns
   Always 1 (this may change in the future)

See void Fl::scheme(const char *name)
remove_check()

```c
void Fl::remove_check (  
    Fl_Timeout_Handler cb,  
    void * argp = 0 ) [static]
```

Removes a check callback.
It is harmless to remove a check callback that no longer exists.

remove_timeout()

```c
void Fl::remove_timeout (  
    Fl_Timeout_Handler cb,  
    void * data = 0 ) [static]
```

Removes a timeout callback from the timer queue.
This method removes all matching timeouts, not just the first one. This may change in the future.
If the `data` argument is NULL (the default!) only the callback `cb` must match, i.e. all timer entries with this callback are removed.
It is harmless to remove a timeout callback that no longer exists.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cb</code></td>
<td>Timer callback to be removed (must match)</td>
</tr>
<tr>
<td><code>data</code></td>
<td>Wildcard if NULL (default), must match otherwise</td>
</tr>
</tbody>
</table>

repeat_timeout()

```c
void Fl::repeat_timeout (  
    double time,  
    Fl_Timeout_Handler cb,  
    void * data = 0 ) [static]
```

Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.
You should call this method only inside a timeout callback of the same or a logically related timer from whose expiration time the new timeout shall be scheduled. Otherwise the timing accuracy can't be improved and the exact behavior is undefined.
If you call this outside a timeout callback the behavior is the same as `Fl::add_timeout()`.

Example: The following code will print "TICK" each second on stdout with a fair degree of accuracy:

```c
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <stdio.h>

void callback(void *) {  
    printf("TICK\n");  
    Fl::repeat_timeout(1.0, callback); // retrigger timeout  
}

int main() {  
    Fl_Window win(100, 100);  
    win.show();  
    Fl::add_timeout(1.0, callback); // set up first timeout  
    return Fl::run();  
}
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>time</code></td>
<td>delta time in seconds until the timer expires</td>
</tr>
<tr>
<td><code>cb</code></td>
<td>callback function</td>
</tr>
<tr>
<td><code>data</code></td>
<td>optional user data (default: NULL)</td>
</tr>
</tbody>
</table>

run()

```c
int Fl::run ( ) [static]
```

Generated by Doxygen
Calls Fl::wait() repeatedly as long as any windows are displayed. When all the windows are closed it returns zero (supposedly it would return non-zero on any errors, but FLTK calls exit directly for these). A normal program will end main() with return Fl::run();.

Note

Fl::run() and Fl::wait() (but not Fl::wait(double)) both return when all FLTK windows are closed. Therefore, a MacOS FLTK application possessing Fl_Sys_Menu_Bar items able to create new windows and expected to keep running without any open window cannot use these two functions. One solution is to run the event loop as follows:

```c
while (!Fl::program_should_quit()) Fl::wait(1e20);
```

12.7.3.53 scheme()

```c
int Fl::scheme (const char * s) [static]
```

Sets the current widget scheme. NULL will use the scheme defined in the FLTK_SCHEME environment variable or the scheme resource under X11. Otherwise, any of the following schemes can be used:

- "none" - This is the default look-n-feel which resembles old Windows (95/98/Me/NT/2000) and old GTK/KDE
- "base" - This is an alias for "none"
- "plastic" - This scheme is inspired by the Aqua user interface on macOS
- "gtk+" - This scheme is inspired by the Red Hat Bluecurve theme
- "gleam" - This scheme is inspired by the Clearlooks Glossy scheme. (Colin Jones and Edmanuel Torres).
- "oxy" - This is a subset of Dmitrij K's oxy scheme (STR 2675, 3477)

If the given scheme name is unknown, the default scheme will be used. Setting the scheme (name) is case insensitive, but the stored scheme name will always be lowercase and Fl::scheme() will return this lowercase name or NULL if no scheme or the default scheme ("none" or "base") was set.

Parameters

- in s Scheme name of NULL

Returns

Current scheme name or NULL

Return values

- NULL if the scheme has not been set or is the default scheme

See also

- Fl::is_scheme()

12.7.3.54 scrollbar_size() [1/2]

```c
int Fl::scrollbar_size () [static]
```

Gets the default scrollbar size used by Fl_Browser_, Fl_Help_View, Fl_Scroll, and Fl_Text_Display widgets.
Returns

The default size for widget scrollbars, in pixels.

12.7.3.55 scrollbar_size() [2/2]

void Fl::scrollbar_size {
    int W } [static]

Sets the default scrollbar size that is used by the Fl_Browser_, Fl_Help_View, Fl_Scroll, and Fl_Text_Display widgets.

Parameters

in W The new default size for widget scrollbars, in pixels.

12.7.3.56 seconds_between()

double Fl::seconds_between {
    Fl_Timestamp & back,
    Fl_Timestamp & further_back } [static]

Return the time in seconds between two time stamps.

Parameters

in back a previously taken time stamp
in further_back an even earlier time stamp

Returns

elapsed seconds and fractions of a second

See also

Fl::seconds_since(Fl_Timestamp& then)
Fl::now()

12.7.3.57 seconds_since()

double Fl::seconds_since {
    Fl_Timestamp & then } [static]

Return the time in seconds between now and a previously taken time stamp.

Parameters

in then a previously taken time stamp

Returns

elapsed seconds and fractions of a second

See also

Fl::seconds_between(Fl_Timestamp& back, Fl_Timestamp& further_back)
Fl::now()
Fl::distant_past()
12.7.3.58  set_box_color()

```c
void Fl::set_box_color (
    Fl_Color c ) [static]
```

Sets the drawing color for the box that is currently drawn. This method sets the current drawing color `fl_color()` depending on the widget's state to either `c` or `fl_inactive(c)`. It should be used whenever a box background is drawn in the box (type) drawing code instead of calling `fl_color(Fl_Color bg)` with the background color `bg`, usually `Fl_Widget::color()`. This method is only useful inside box drawing code. Whenever a box is drawn with one of the standard box drawing methods, a static variable is set depending on the widget's current state - if the widget is `!active_r()` then the internal variable is false (0), otherwise it is true (1). This is faster than calling `Fl_Widget::active_r()` because the state is cached.

See also

- `Fl::draw_box_active()`
- `Fl::box_color(Fl_Color)`

12.7.3.59  set_idle()

```c
static void Fl::set_idle (
    Fl_Old_Idle_Handler cb ) [inline], [static]
```

Sets an idle callback.

**Deprecated** This method is obsolete - use the `add_idle()` method instead.

12.7.3.60  ticks_between()

```c
long Fl::ticks_between (
    Fl_Timestamp & back,
    Fl_Timestamp & further_back ) [static]
```

Return the time in ticks (60Hz) between two time stamps.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>back</th>
<th>a previously taken time stamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>further_back</td>
<td>an even earlier time stamp</td>
</tr>
</tbody>
</table>

**Returns**

elapsed ticks in 60th of a second

See also

- `Fl::ticks_since(Fl_Timestamp& then)`
- `Fl::now()`

12.7.3.61  ticks_since()

```c
long Fl::ticks_since (
    Fl_Timestamp & then ) [static]
```

Return the time in ticks (60Hz) between now and a previously taken time stamp. Ticks are a convenient way to time animations 'per frame'. Even though modern computers use all kinds of screen refresh rates, 60Hz is a very good base for animation that is typically shown in user interface graphics.

**Parameters**

| in  | then         | a previously taken time stamp |

Generated by Doxygen
Class Documentation

Returns

elapsed ticks in 60th of a second

See also

Fl::ticks_between(Fl_Timestamp& back, Fl_Timestamp& further_back)
Fl::now()

12.7.3.62  use_high_res_GL() [1/2]

static int Fl::use_high_res_GL ( ) [inline], [static]
returns whether GL windows should be drawn at high resolution on Apple computers with retina displays.
Default is no.
Version

  1.3.4

12.7.3.63  use_high_res_GL() [2/2]

static void Fl::use_high_res_GL ( int val ) [inline], [static]
sets whether GL windows should be drawn at high resolution on Apple computers with retina displays
Version

  1.3.4

12.7.3.64  version()

double Fl::version ( ) [static]
Returns the compiled-in value of the FL_VERSION constant.
This is useful for checking the version of a shared library.

Deprecated  Use int Fl::api_version() instead.

12.7.3.65  visible_focus() [1/2]

static int Fl::visible_focus ( ) [inline], [static]
Gets or sets the visible keyboard focus on buttons and other non-text widgets.
The default mode is to enable keyboard focus for all widgets.

12.7.3.66  visible_focus() [2/2]

static void Fl::visible_focus ( int v ) [inline], [static]
Gets or sets the visible keyboard focus on buttons and other non-text widgets.
The default mode is to enable keyboard focus for all widgets.

12.7.3.67  visual()

int Fl::visual ( int flags ) [static]
Selects a visual so that your graphics are drawn correctly.
This is only allowed before you call show() on any windows. This does nothing if the default visual satisfies the
capabilities, or if no visual satisfies the capabilities, or on systems that don’t have such brain-dead notions.
Only the following combinations do anything useful:
12.7 Fl Class Reference

- Fl::visual(FL_RGB)
  Full true color (if there are several depths FLTK chooses the largest). Do this if you use fl_draw_image for much better (non-dithered) output.

- Fl::visual(FL_RGB8)
  Full color with at least 24 bits of color. FL_RGB will always pick this if available, but if not it will happily return a less-than-24 bit deep visual. This call fails if 24 bits are not available.

This returns true if the system has the capabilities by default or FLTK succeeded in turning them on. Your program will still work even if this returns false (it just won’t look as good).

12.7.3.68 wait() [1/2]

```c
int Fl::wait () [static]
```

Waits until "something happens" and then returns.
Call this repeatedly to "run" your program. You can also check what happened each time after this returns, which is quite useful for managing program state.
What this really does is call all idle callbacks, all elapsed timeouts, call Fl::flush() to get the screen to update, and then wait some time (zero if there are idle callbacks, the shortest of all pending timeouts, or infinity), for any events from the user or any Fl::add_fd() callbacks. It then handles the events and calls the callbacks and then returns.

Returns
non-zero if there are any visible windows - this may change in future versions of FLTK.

12.7.3.69 wait() [2/2]

```c
double Fl::wait ( double time_to_wait ) [static]
```

Waits a maximum of time_to_wait seconds or until "something happens".
See Fl::wait() for the description of operations performed when "something happens".

Returns
Always 1 on Windows. Otherwise, it is positive if an event or fd happens before the time elapsed. It is zero if nothing happens. It is negative if an error occurs (this will happen on X11 if a signal happens).

12.7.4 Member Data Documentation

12.7.4.1 help

```c
const char *const Fl::help = helpmsg+13 [static]
```

Usage string displayed if Fl::args() detects an invalid argument.
This may be changed to point to customized text at run-time.

12.7.4.2 idle

```c
void(* Fl::idle)() [static]
```

The currently executing idle callback function: DO NOT USE THIS DIRECTLY!
This is now used as part of a higher level system allowing multiple idle callback functions to be called.

See also
add_idle(), remove_idle()

The documentation for this class was generated from the following files:

- Fl.H
- Fl.cxx
- Fl_abort.cxx

Generated by Doxygen
12.8 Fl_Adjuster Class Reference

The Fl_Adjuster widget was stolen from Prisms, and has proven to be very useful for values that need a large dynamic range.

```cpp
#include <Fl_Adjuster.H>
```

Inheritance diagram for Fl_Adjuster:

```
Fl_Widget
    ↓
Fl_Valuator
    ↓
Fl_Adjuster
```

Public Member Functions

- **Fl_Adjuster** (int X, int Y, int W, int H, const char ∗l=0)
  
  Creates a new Fl_Adjuster widget using the given position, size, and label string.

- **int soft () const**
  
  If "soft" is turned on, the user is allowed to drag the value outside the range.

- **void soft (int s)**
  
  If "soft" is turned on, the user is allowed to drag the value outside the range.

Public Member Functions inherited from Fl_Valuator

- **void bounds** (double a, double b)
  
  Sets the minimum (a) and maximum (b) values for the valuator widget.

- **double clamp** (double)
  
  Clamps the passed value to the valuator range.

- **virtual int format** (char ∗)
  
  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.

- **double increment** (double, int)
Adds \( n \) times the step value to the passed value.

- **double maximum () const**
  
  Gets the maximum value for the valuator.

- **void maximum (double a)**
  
  Sets the maximum value for the valuator.

- **double minimum () const**
  
  Gets the minimum value for the valuator.

- **void minimum (double a)**
  
  Sets the minimum value for the valuator.

- **void precision (int digits)**
  
  Sets the step value to \( 1.0 / 10^{digits} \).

- **void range (double a, double b)**
  
  Sets the minimum and maximum values for the valuator.

- **double round (double)**
  
  Round the passed value to the nearest step increment.

- **double step () const**
  
  Gets or sets the step value.

- **void step (double a, int b)**
  
  See double Fl_Valuator::step() const

- **void step (double s)**
  
  See double Fl_Valuator::step() const.

- **void step (int a)**
  
  See double Fl_Valuator::step() const

- **double value () const**
  
  Gets the floating point(double) value.

- **int value (double)**
  
  Sets the current value.

### Public Member Functions inherited from Fl_Widget

- **void _clear_fullscreen ()**
  
  Activates the widget.

- **void _set_fullscreen ()**
  
  Activates the widget.

- **unsigned int active () const**
  
  Returns whether the widget is active.

- **int active_r () const**
  
  Returns whether the widget and all of its parents are active.

- **Fl_Align align () const**
  
  Gets the label alignment.

- **void align (Fl_Align alignment)**
  
  Sets the label alignment.

- **long argument () const**
  
  Gets the current user data (long) argument that is passed to the callback function.

- **void argument (long v)**
  
  Sets the current user data (long) argument that is passed to the callback function.

- **virtual class Fl_Gl_Window * as_gl_window ()**
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- **virtual Fl_Gl_Window const * as_gl_window () const**

- **virtual Fl_Group * as_group ()**

- **virtual Fl_Group const * as_group () const**
• virtual Fl_Group const & as_group () const
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Window * & as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• void bind_deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

• void bind_image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.

• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

• Fl_Boxtype box () const
  Gets the box type of the widget.

• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.

• Fl_Callback_p callback () const
  Gets the current callback function for the widget.

• void callback (Fl_Callback *cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

• void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.

• void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.

• unsigned int changed () const
  Checks if the widget value changed since the last callback.

• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

• void clear_changed ()
  Marks the value of the widget as unchanged.

• void clear_damage (uchar c=0)
  Clears or sets the damage flags.

• void clear_output ()
  Sets a widget to accept input.

• void clear_visible ()
  Hides the widget.

• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

• Fl_Color color () const
  Gets the background color of the widget.

• void color (Fl_Color bg)
  Sets the background color of the widget.

• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget ∗w) const
  Checks if w is a child of this widget.
• void copy_label (const char ∗new_label)
  Sets the current label.
• void copy_tooltip (const char ∗text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image ∗deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image ∗deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget ∗widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget ∗widget, void ∗arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image ∗image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image ∗image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
Checks if this widget is a child of wgt.

- `int is_label_copied () const`  
  `Returns whether the current label was assigned with copy_label().`

- `const char ∗ label () const`  
  `Gets the current label text.`

- `void label (const char ∗text)`  
  `Sets the current label pointer.`

- `void label (Fl_Labeltype a, const char ∗b)`  
  `Shortcut to set the label text and type in one call.`

- `Fl_Color labelcolor () const`  
  `Gets the label color.`

- `void labelcolor (Fl_Color c)`  
  `Sets the label color.`

- `Fl_Font labelfont () const`  
  `Gets the font to use.`

- `void labelfont (Fl_Font f)`  
  `Sets the font to use.`

- `Fl_Fontsize labelsize () const`  
  `Gets the font size in pixels.`

- `void labelsize (Fl_Fontsize pix)`  
  `Sets the font size in pixels.`

- `Fl_Labeltype labeltype () const`  
  `Gets the label type.`

- `void labeltype (Fl_Labeltype a)`  
  `Sets the label type.`

- `void measure_label (int &ww, int &hh) const`  
  `Sets width ww and height hh accordingly with the label size.`

- `bool needs_keyboard () const`  
  `Returns whether this widget needs a keyboard.`

- `void needs_keyboard (bool needs)`  
  `Sets whether this widget needs a keyboard.`

- `unsigned int output () const`  
  `Returns if a widget is used for output only.`

- `Fl_Group ∗ parent () const`  
  `Returns a pointer to the parent widget.`

- `void parent (Fl_Group ∗p)`  
  `Internal use only - "for hacks only".`

- `void position (int X, int Y)`  
  `Repositions the window or widget.`

- `void redraw ()`  
  `Schedules the drawing of the widget.`

- `void redraw_label ()`  
  `Schedules the drawing of the label.`

- `virtual void resize (int x, int y, int w, int h)`  
  `Changes the size or position of the widget.`

- `Fl_Color selection_color () const`  
  `Gets the selection color.`

- `void selection_color (Fl_Color a)`  
  `Sets the selection color.`

- `void set_active ()`  
  `Marks the widget as active without sending events or changing focus.`
12.8 Fl_Adjuster Class Reference

- void set_changed ()
  Marks the value of the widget as changed.
- void set_output ()
  Sets a widget to output only.
- void set_visible ()
  Makes the widget visible.
- void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
- int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()
  Makes a widget visible.
- void size (int W, int H)
  Changes the size of the widget.
- int take_focus ()
  Gives the widget the keyboard focus.
- unsigned int takeevents () const
  Returns if the widget is able to take events.
- int test_shortcut ()
  Returns true if the widget's label contains the entered 'x' shortcut.
- const char ∗ tooltip () const
  Gets the current tooltip text.
- void tooltip (const char ∗text)
  Sets the current tooltip text.
- Fl_Window ∗ top_window () const
  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const
  Gets the widget type.
- void type (uchar t)
  Sets the widget type.
- int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const
  Gets the user data for this widget.
- void user_data (Fl_Callback ∗v, bool auto_free)
  Sets the user data for this widget.
- void user_data (void ∗v)
  Sets the user data for this widget.
- unsigned int visible () const
  Returns whether a widget is visible.
- unsigned int visible_focus () const
  Checks if this widget has a visible focus.
- void visible_focus (int v)
  Modifies keyboard focus navigation.
- int visible_r () const
  Returns whether a widget and all its parents are visible.
- int w () const
Gets the widget width.

• **FL_When when ()** const
  Returns the conditions under which the callback is called.

• **void when (uchar i)**
  Sets the flags used to decide when a callback is called.

• **FL_Window * window ()** const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• **int x ()** const
  Gets the widget position in its window.

• **int y ()** const
  Gets the widget position in its window.

• **virtual ~FL_Widget ()**
  Destroys the widget.

**Protected Member Functions**

• **void draw ()** FL_OVERRIDE
  Draws the widget.

• **int handle (int)** FL_OVERRIDE
  Handles the specified event.

• **void value_damage ()** FL_OVERRIDE
  Asks for partial redraw.

**Protected Member Functions inherited from FL_Valuator**

• **FL_Valuator (int X, int Y , int W, int H, const char ∗L)**
  Creates a new FL_Valuator widget using the given position, size, and label string.

• **void handle_drag (double newvalue)**
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.

• **void handle_push ()**
  Stores the current value in the previous value.

• **void handle_release ()**
  Called after an FL_WHEN_RELEASE event is received and before the callback.

• **int horizontal ()** const
  Tells if the valuator is an FL_HORIZONTAL one.

• **double previous_value ()** const
  Gets the previous floating point value before an event changed it.

• **void set_value (double v)**
  Sets the current floating point value.

• **double softclamp (double)**
  Clamps the value, but accepts v if the previous value is not already out of range.

**Protected Member Functions inherited from FL_Widget**

• **void clear_flag (unsigned int c)**
  Clears a flag in the flags mask.

• **void draw_backdrop ()** const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• **void draw_box ()** const
  Draws the widget box according its box style.

• **void draw_box (FL_Boxtype t, FL_Color c)** const
  Draws a box of type t, of color c at the widget's position and size.
• void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type \( t \), of color \( c \) at the position \( X,Y \) and size \( W,H \).

• void **draw_focus** () const
  
  Draws a focus rectangle around the widget.

• void **draw_focus** (Fl_Boxtype t, int x, int y, int w, int h) const
  
  Draws a focus rectangle around the widget.

• void **draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.

• void **draw_label** () const
  
  Draws the widget's label at the defined label position.

• void **draw_label** (int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.

• Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  
  Creates a widget at the given position and size.

• unsigned int **flags** () const
  
  Gets the widget flags mask.

• void **h** (int v)
  
  Internal use only.

• void **set_flag** (unsigned int c)
  
  Sets a flag in the flags mask.

• void **w** (int v)
  
  Internal use only.

• void **x** (int v)
  
  Internal use only.

• void **y** (int v)
  
  Internal use only.

### Additional Inherited Members

**Static Public Member Functions inherited from Fl_Widget**

• static void **default_callback** (Fl_Widget ∗widget, void ∗data)
  
  The default callback for all widgets that don't set a callback.

• static unsigned int **label_shortcut** (const char ∗t)
  
  Returns the Unicode value of the '&x' shortcut in a given text.

• static int **test_shortcut** (const char ∗t, const bool require_alt=false)
  
  Returns true if the given text \( t \) contains the entered '&x' shortcut.

**Protected Types inherited from Fl_Widget**

• enum {
  
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
  }

  flags possible values enumeration.
12.8.1 Detailed Description

The Fl_Adjuster widget was stolen from Prisms, and has proven to be very useful for values that need a large dynamic range.

![Fl_Adjuster widget](image)

Figure 12.1 Fl_Adjuster

When you press a button and drag to the right the value increases. When you drag to the left it decreases. The largest button adjusts by $100 \times \text{step}()$, the next by $10 \times \text{step}()$ and that smallest button by $\text{step}()$. Clicking on the buttons increments by 10 times the amount dragging by a pixel does. Shift + click decrements by 10 times the amount.

12.8.2 Constructor & Destructor Documentation

12.8.2.1 Fl_Adjuster()

```cpp
Fl_Adjuster::Fl_Adjuster ( int X, int Y, int W, int H, const char * l = 0 )
```

Creates a new Fl_Adjuster widget using the given position, size, and label string. It looks best if one of the dimensions is 3 times the other. Inherited destructor destroys the Valuator.

12.8.3 Member Function Documentation

12.8.3.1 draw()

```cpp
void Fl_Adjuster::draw ( ) [protected], [virtual]
```

Draws the widget. Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead. Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

12.8.3.2 handle()

```cpp
int Fl_Adjuster::handle ( int event ) [protected], [virtual]
```

Handles the specified event. You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class. In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>event</th>
<th>the kind of event received</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th></th>
<th>if the event was not used or understood</th>
<th>if the event was used and can be deleted</th>
</tr>
</thead>
</table>

See also

Fl_Event

Reimplemented from Fl_Widget.

12.8.3.3 soft() [1/2]

int Fl_Adjuster::soft ( ) const [inline]

If "soft" is turned on, the user is allowed to drag the value outside the range. If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value. Default is one.

12.8.3.4 soft() [2/2]

void Fl_Adjuster::soft ( int s ) [inline]

If "soft" is turned on, the user is allowed to drag the value outside the range. If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value. Default is one.

12.8.3.5 value_damage()

void Fl_Adjuster::value_damage ( ) [protected], [virtual]

Asks for partial redraw.

Reimplemented from Fl_Valuator.

The documentation for this class was generated from the following files:

- Fl_Adjuster.H
- Fl_Adjuster.cxx

12.9 Fl_Anim_GIF_Image Class Reference

The Fl_Anim_GIF_Image class supports loading, caching, and drawing of animated Compuserve GIF\textsuperscript{SM} images. Inheritance diagram for Fl_Anim_GIF_Image:
Public Types

- enum Flags {
  DONT_START = 1, DONT_RESIZE_CANVAS = 2, DONT_SET_AS_IMAGE = 4, OPTIMIZE_MEMORY = 8,
  LOG_FLAG = 64, DEBUG_FLAG = 128 }

  When opening an Fl_Anim_GIF_Image there are some options that can be passed in a flags value.

Public Member Functions

- Fl_Widget * canvas() const
  Gets the current widget, that is used to display the frame images.

- void canvas(Fl_Widget *canvas, unsigned short flags=0)
  Link the image back to a widget for automated animation.

- int canvas_h() const
  Return the height of the animation canvas.

- int canvas_w() const
  Return the width of the animation canvas.

- void color_average(Fl_Color c, float i) FL_OVERRIDE
  Applies a color average to all frames.

- Fl_Image * copy() const
  Copy and resize the animation frames.

- Fl_Image * copy(int W, int H) const FL_OVERRIDE
  Copy and resize the animation frames.

- int debug() const

- void delay(int frame, double delay)
  Set the delay of frame [0-frames()-1] in seconds.

- double delay(int frame_) const
  Return the delay of frame [0-frames()-1] in seconds.

- void desaturate() FL_OVERRIDE
  Desaturate to all frames of the animation.

- void draw(int x, int y, int w, int h, int cx=0, int cy=0) FL_OVERRIDE
  Draw the current frame of the animation.

- Fl_Anim_GIF_Image()
  Create an empty animated GIF image shell.

- Fl_Anim_GIF_Image(const char *filename, Fl_Widget *canvas=0, unsigned short flags=0)
  Load an animated GIF image from a file.

- Fl_Anim_GIF_Image(const char *imagename, const unsigned char *data, const size_t length, Fl_Widget *canvas=0, unsigned short flags=0)
  Load an animated GIF image from memory.

- int frame() const
  Return the current frame.

- void frame(int frame)
Set the current frame.

- `int frame_h(int frame) const`
  Return the frame dimensions of a frame.

- `bool frame_uncache() const`
  Return the active frame_uncache() setting.

- `void frame_uncache(bool uncache)`
  Use frame_uncache() to set or forbid frame image uncaching.

- `int frame_w(int frame) const`
  Return the frame dimensions of a frame.

- `int frame_x(int frame) const`
  Return the frame position of a frame.

- `int frame_y(int frame) const`
  Return the frame position of a frame.

- `int frames() const`
  Get the number of frames in the animation.

- `Fl_Image * image() const`
  Return the current frame image.

- `Fl_Image * image(int frame) const`
  Return the image of the given frame index.

- `bool is_animated() const`
  Check if this is a valid animation with more than one frame.

- `bool load(const char *name, const unsigned char *imgdata=NULL, size_t imglength=0)`
  Load an animation from a file or from a memory block.

- `const char * name() const`
  Return the name of the played file as specified in the constructor.

- `bool next()`
  Show the next frame if the animation is stopped.

- `bool playing() const`
  Return if the animation is currently running or stopped.

- `Fl_Anim_GIF_Image & resize(double scale)`
  Resizes the image to the specified size, replacing the current image.

- `Fl_Anim_GIF_Image & resize(int w, int h)`
  Resizes the image to the specified size, replacing the current image.

- `double speed() const`
  Get the animation speed factor.

- `void speed(double speed)`
  Set the animation speed factor.

- `bool start()`
  The start() method (re-)starts the playing of the frames.

- `bool stop()`
  The stop() method stops the playing of the frames.

- `void uncache() FL_OVERRIDE`
  Uncache all cached image data now.

- `bool valid() const`
  Check if animation is valid.

- `~Fl_Anim_GIF_Image() FL_OVERRIDE`
  Release the image and all cached data.
Public Member Functions inherited from Fl_GIF_Image

- **Fl_GIF_Image (const char *filename)**
  This constructor loads a GIF image from the given file.

- **Fl_GIF_Image (const char *imagename, const unsigned char *data)**
  This constructor loads a GIF image from memory (deprecated).

- **Fl_GIF_Image (const char *imagename, const unsigned char *data, const size_t length)**
  This constructor loads a GIF image from memory.

Public Member Functions inherited from Fl_Pixmap

- **int cache_h ()**
- **int cache_w ()**

- **void color_average (Fl_Color c, float i) FL_OVERRIDE**
  The color_average() method averages the colors in the image with the provided FLTK color value.

- **Fl_Pixmap (char *const *D)**
  The constructors create a new pixmap from the specified XPM data.

- **Fl_Pixmap (const char *const *D)**
  The constructors create a new pixmap from the specified XPM data.

- **Fl_Pixmap (const uchar *const *D)**
  The constructors create a new pixmap from the specified XPM data.

- **Fl_Pixmap (uchar *const *D)**
  The constructors create a new pixmap from the specified XPM data.

- **void label (Fl_Menu_Item *m) FL_OVERRIDE**
  This method is an obsolete way to set the image attribute of a menu item.

- **void label (Fl_Widget *w) FL_OVERRIDE**
  This method is an obsolete way to set the image attribute of a widget or menu item.

- **void uncache () FL_OVERRIDE**
  If the image has been cached for display, delete the cache data.

- **virtual ~Fl_Pixmap ()**
  The destructor frees all memory and server resources that are used by the pixmap.

Public Member Functions inherited from Fl_Image

- **virtual class Fl_Shared_Image * as_shared_image ()**
  Returns whether an image is an Fl_Shared_Image or not.

- **Fl_Pixmap (int X, int Y)**
  Draws the image to the current drawing surface with a bounding box.

- **void draw ()**
  The constructors create a new pixmap from the specified XPM data.

- **void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE**
  Creates a resized copy of the image.

- **void desaturate () FL_OVERRIDE**
  The desaturate() method converts an image to grayscale.

- **int count () const**
  Returns the number of data values associated with the image.

- **int d () const**
  Returns the image depth.

- **const char *const *data () const**
  Returns a pointer to the current image data array.
• int \texttt{data\_h} () const
  Returns the height of the image data.
• int \texttt{data\_w} () const
  Returns the width of the image data.
• void \texttt{draw}(int X, int Y)
  Draws the image to the current drawing surface.
• int \texttt{fail} () const
  Returns a value that is not 0 if there is currently no image available.
• \texttt{Fl\_Image}(int W, int H, int D)
  The constructor creates an empty image with the specified width, height, and depth.
• int \texttt{h} () const
  Returns the current image drawing height in FLTK units.
• void \texttt{inactive}()
  The \texttt{inactive()} method calls \texttt{color\_average(FL\_BACKGROUND\_COLOR, 0.33f)} to produce an image that appears grayed out.
• int \texttt{id} () const
  Returns the current line data size in bytes.
• virtual void \texttt{release}() 
  Releases an \texttt{Fl\_Image} - the same as \texttt{delete this}.
• virtual void \texttt{scale}(int width, int height, int proportional=1, int can\_expand=0)
  Sets the drawing size of the image.
• int \texttt{w} () const
  Returns the current image drawing width in FLTK units.
• virtual \texttt{\sim Fl\_Image}()
  The destructor is a virtual method that frees all memory used by the image.

Static Public Member Functions

• static int \texttt{frame\_count}(const char *name, const unsigned char *imgdata=NULL, size_t imglength=0)
  Get the number of frames in a GIF file or in a GIF compressed data block.

Static Public Member Functions inherited from \texttt{Fl\_GIF\_Image}

• static bool \texttt{is\_animated}(const char *name_)

Static Public Member Functions inherited from \texttt{Fl\_Image}

• static \texttt{Fl\_Labeltype\ define\_FL\_IMAGE\_LABEL}()
• static \texttt{Fl\_RGB\_Scaling\ RGB\_scaling}()
  Returns the currently used RGB image scaling method.
• static void \texttt{RGB\_scaling}(\texttt{Fl\_RGB\_Scaling})
  Sets the RGB image scaling method used for copy(int, int).
• static \texttt{Fl\_RGB\_Scaling\ scaling\_algorithm}()
  Gets what algorithm is used when resizing a source image to draw it.
• static void \texttt{scaling\_algorithm}(\texttt{Fl\_RGB\_Scaling} algorithm)
  Sets what algorithm is used when resizing a source image to draw it.

Static Public Attributes

• static bool \texttt{loop} = true
  The loop flag can be used to (dis-)allow loop count.
• static double \texttt{min\_delay} = 0.
  The \texttt{min\_delay} value can be used to set a minimum value for the frame delay for playback.
Static Public Attributes inherited from Fl_GIF_Image

- static bool animate = false
  
  Sets how the shared image core routine should treat animated GIF files.

Static Public Attributes inherited from Fl_Image

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_MEMORY_ACCESS = -4
- static const int ERR_NO_IMAGE = -1
- static bool register_images_done = false
  
  True after fl_register_images() was called, false before.

Protected Member Functions

- void clear_frames ()
- bool next_frame ()
- void on_extension_data (Fl_GIF_Image::GIF_FRAME &f) FL_OVERRIDE
- void on_frame_data (Fl_GIF_Image::GIF_FRAME &f) FL_OVERRIDE
- void scale_frame ()
- void set_frame ()
- void set_frame (int frame)

Protected Member Functions inherited from Fl_GIF_Image

- Fl_GIF_Image ()
  
  The default constructor creates an empty GIF image.

- Fl_GIF_Image (const char ∗ filename, bool anim)
- Fl_GIF_Image (const char ∗ imagename, const unsigned char ∗ data, const size_t length, bool anim)
  
  The protected load() methods are used by Fl_Anim_GIF_Image to request loading of animated GIF's.

- void load (const char ∗ imagename, const unsigned char ∗ data, const size_t length, bool anim)
- void load_gif_ (class Fl_Image_Reader &rdr, bool anim=false)

Protected Member Functions inherited from Fl_Pixmap

- void measure ()

Protected Member Functions inherited from Fl_Image

- void d (int D)
  
  Sets the current image depth.

- void data (const char ∗ ∗ p, int c)
  
  Sets the current data pointer and count of pointers in the array.

- void draw_empty (int X, int Y)
  
  The protected draw_empty() draws a box with an X in it.

- int draw_scaled (int X, int Y, int W, int H)
  
  Draw the image to the current drawing surface rescaled to a given width and height.

- void h (int H)
  
  Sets the height of the image data.

- void Id (int LD)
  
  Sets the current line data size in bytes.

- void w (int W)
  
  Sets the width of the image data.
12.9 Fl_Anim_GIF_Image Class Reference

Static Protected Member Functions

• static void cb_animate (void *d)

Static Protected Member Functions inherited from Fl_Image

• static void labeltype (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
• static void measure (const Fl_Label *lo, int &lw, int &lh)

Additional Inherited Members

Public Attributes inherited from Fl_Pixmap

• int alloc_data

12.9.1 Detailed Description

The Fl_Anim_GIF_Image class supports loading, caching, and drawing of animated Compuserve GIF\textsuperscript{SM} images. The class loads all images contained in the file and animates them by cycling through them as defined by the delay times in the image file. The user must supply an FLTK widget as "container" in order to see the animation by specifying it in the constructor or later using the canvas() method.

12.9.2 Member Enumeration Documentation

12.9.2.1 Flags

enum Fl_Anim_GIF_Image::Flags

When opening an Fl_Anim_GIF_Image there are some options that can be passed in a flags value.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DONT_START</td>
<td>This flag indicates to the loader that it should not start the animation immediately after successful load, which is the default. It can be started later using the start() method.</td>
</tr>
<tr>
<td>DONT_RESIZE_CANVAS</td>
<td>This flag indicates to the loader that it should not resize the canvas widget of the animation to the dimensions of the animation, which is the default. Needed for special use cases.</td>
</tr>
<tr>
<td>DONT_SET_AS_IMAGE</td>
<td>This flag indicates to the loader that it should not set the animation as image() member of the canvas widget, which is the default. Needed for special use cases.</td>
</tr>
<tr>
<td>OPTIMIZE_MEMORY</td>
<td>Often frames change just a small area of the animation canvas. This flag indicates to the loader to try using less memory by storing frame data not as canvas-sized images but use the sizes defined in the GIF file. The drawbacks are higher cpu usage during playback and maybe minor artifacts when resized.</td>
</tr>
<tr>
<td>LOG_FLAG</td>
<td>This flag can be used to print informations about the decoding process to the console.</td>
</tr>
<tr>
<td>DEBUG_FLAG</td>
<td>This flag can be used to print even more informations about the decoding process to the console.</td>
</tr>
</tbody>
</table>

12.9.3 Constructor & Destructor Documentation

12.9.3.1 Fl_Anim_GIF_Image() [1/2]

Fl_Anim_GIF_Image::Fl_Anim_GIF_Image ( const char * filename, Fl_Widget * canvas = 0, unsigned short flags = 0 )

Load an animated GIF image from a file.
This constructor creates an animated image from a GIF-formatted file. Optionally it applies the \texttt{canvas()} method after successful load. If \texttt{DONT\_START} is not specified in the \texttt{flags} parameter it calls \texttt{start()} after successful load.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{filename}</td>
<td>path and name of GIF file in the file system</td>
</tr>
<tr>
<td>\texttt{canvas}</td>
<td>a widget that will show and animate the GIF, or NULL</td>
</tr>
<tr>
<td>\texttt{flags}</td>
<td>see Flags for details, or 0</td>
</tr>
</tbody>
</table>

Note

The GIF image must be decoupled from the canvas by calling \texttt{myGif->canvas(NULL)}; before deleting the canvas.

12.9.3.2 \texttt{Fl\_Anim\_GIF\_Image()} [2/2]

\texttt{Fl\_Anim\_GIF\_Image::Fl\_Anim\_GIF\_Image ( const char * \texttt{imagename},
const unsigned char * \texttt{data},
const size_t \texttt{length},
Fl\_Widget * \texttt{canvas} = 0,
unsigned short \texttt{flags} = 0 )}

Load an animated GIF image from memory.
This constructor creates an animated image from a GIF-formatted block in memory. Optionally it applies the \texttt{canvas()} method after successful load. If \texttt{DONT\_START} is not specified in the \texttt{flags} parameter it calls \texttt{start()} after successful load.
\texttt{imagename} can be NULL. If a name is given, the image is added to the list of shared images and will be available by that name.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{imagename}</td>
<td>a name given to this image or NULL</td>
</tr>
<tr>
<td>\texttt{data}</td>
<td>pointer to the start of the GIF image in memory</td>
</tr>
<tr>
<td>\texttt{length}</td>
<td>length of the GIF image in memory</td>
</tr>
<tr>
<td>\texttt{canvas}</td>
<td>a widget that will show and animate the GIF, or NULL</td>
</tr>
<tr>
<td>\texttt{flags}</td>
<td>see Flags for details, or 0</td>
</tr>
</tbody>
</table>

Note

The GIF image must be decoupled from the canvas by calling \texttt{myGif->canvas(NULL)}; before deleting the canvas.

12.9.3.3 \texttt{\~{}Fl\_Anim\_GIF\_Image()}

\texttt{Fl\_Anim\_GIF\_Image::\~{}Fl\_Anim\_GIF\_Image ( )}

Release the image and all cached data.
Also removes the animation timer.

12.9.4 Member Function Documentation

12.9.4.1 \texttt{canvas()} [1/2]

\texttt{Fl\_Widget * Fl\_Anim\_GIF\_Image::canvas ( ) const}

Gets the current widget, that is used to display the frame images.
Returns

a pointer to a widget

### 12.9.4.2 canvas() [2/2]

```cpp
void Fl_Anim_GIF_Image::canvas (  
    Fl_Widget * canvas,
    unsigned short flags = 0 )
```

Link the image back to a widget for automated animation.

This method sets current widget, that is used to display the frame images. The `flags` parameter specifies whether the canvas widget is resized to the animation dimensions and/or its `image()` method will be used to set the current frame image during animation.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>canvas</th>
<th>a pointer to the widget that will show the animation</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>flags</td>
<td>see Flags</td>
</tr>
</tbody>
</table>

**Note**

The GIF image must be decoupled from the canvas by calling `myGif->canvas(NULL);` before deleting the canvas.

### 12.9.4.3 canvas_h()

```cpp
int Fl_Anim_GIF_Image::canvas_h ( ) const
```

Return the height of the animation canvas.

**Returns**

the width in pixel units

### 12.9.4.4 canvas_w()

```cpp
int Fl_Anim_GIF_Image::canvas_w ( ) const
```

Return the width of the animation canvas.

**Returns**

the width in pixel units

### 12.9.4.5 color_average()

```cpp
void Fl_Anim_GIF_Image::color_average (  
    Fl_Color c,  
    float i ) [virtual]
```

Applies a color average to all frames.

The `color_average()` method averages the colors in the image with the provided FLTK color value.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>c</th>
<th>blend color</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i</td>
<td>a value between 0.0 and 1.0 where 0 results in the blend color, and 1 returns the original image</td>
</tr>
</tbody>
</table>

Reimplemented from `Fl_Image`. 

Generated by Doxygen
12.9.4.6  copy()

```cpp
Fl_Image * Fl_Anim_GIF_Image::copy (  
  int W,  
  int H ) const [virtual]
```

Copy and resize the animation frames.
The virtual copy() method makes a copy of the animated image and resizes all of its frame images to \( W \times H \) using the current resize method.

**Parameters**

| in  | W,H | new size in FLTK pixel units |

**Returns**

the resized copy of the animation

Reimplemented from Fl_Image.

12.9.4.7  delay() [1/2]

```cpp
void Fl_Anim_GIF_Image::delay (  
  int frame,  
  double delay )
```

Set the delay of frame \([0:\text{frames} -1]\) in seconds.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>frame</th>
<th>index into frame list</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>delay</td>
<td>to next frame in seconds</td>
</tr>
</tbody>
</table>

12.9.4.8  delay() [2/2]

```cpp
double Fl_Anim_GIF_Image::delay (  
  int frame ) const
```

Return the delay of frame \([0:\text{frames} -1]\) in seconds.

**Parameters**

| in  | frame | index into frame list |

**Returns**

delay to next frame in seconds

12.9.4.9  desaturate()

```cpp
void Fl_Anim_GIF_Image::desaturate ( ) [virtual]
```

Desaturate to all frames of the animation.
Reimplemented from Fl_Image.

12.9.4.10  draw()

```cpp
void Fl_Anim_GIF_Image::draw (  
  int x,  
  int y,  
  int w,
```
12.9 Fl_Anim_GIF_Image Class Reference

Draw the current frame of the animation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x,y,w,h</th>
<th>target rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cx, cy</td>
<td>source offset</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Image.

12.9.4.11 frame() [1/2]

```cpp
int Fl_Anim_GIF_Image::frame ( ) const
```

Return the current frame.

Returns

the current frame index in the range for 0 to frames()-1.
-1 if the image has no frames.

12.9.4.12 frame() [2/2]

```cpp
void Fl_Anim_GIF_Image::frame ( int frame )
```

Set the current frame.

Parameters

| in   | frame | index into list of frames |

12.9.4.13 frame_count()

```cpp
int Fl_Anim_GIF_Image::frame_count ( const char * name, const unsigned char * imgdata = NULL, size_t imglength = 0 ) [static]
```

Get the number of frames in a GIF file or in a GIF compressed data block.

The static frame_count() method is just a convenience method for getting the number of images (frames) stored in a GIF file.

As this count is not readily available in the GIF header, the whole GIF file has be parsed (which is done here by using a temporary Fl_Anim_GIF_Image object for simplicity). So this call may be slow with large files.

If imgdata is NULL, the image will be read from the file. Otherwise, it will be read from memory.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>path and name of GIF file in the file system, ignored when reading from memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>imgdata</td>
<td>pointer to the start of the GIF image in memory, or NULL to read from a file</td>
</tr>
<tr>
<td>in</td>
<td>imglength</td>
<td>length of the GIF image in memory, or 0</td>
</tr>
</tbody>
</table>
Returns
the number of frames in the animation

12.9.4.14  frame_h()

int Fl_Anim_GIF_Image::frame_h (  
    int frame ) const

Return the frame dimensions of a frame.
Usefull only if loaded with 'optimize_mem' and the animation also has size optimized frames.

Parameters

    in frame index into frame list

Returns

height in FLTK pixle units

12.9.4.15  frame_uncache () [1/2]

bool Fl_Anim_GIF_Image::frame_uncache ( ) const

Return the active frame_uncache() setting.

Returns

true if caching is disabled

12.9.4.16  frame_uncache () [2/2]

void Fl_Anim_GIF_Image::frame_uncache (  
    bool uncache )

Use frame_uncache() to set or forbid frame image uncaching.
If frame uncaching is set, frame images are not offscreen cached for re-use and will be re-created every time they
are displayed. This saves a lot of memory on the expense of cpu usage and should be carefully considered. Per
default frame caching will be done.

Parameters

    in uncache true to disable caching

12.9.4.17  frame_w()

int Fl_Anim_GIF_Image::frame_w (  
    int frame ) const

Return the frame dimensions of a frame.
Usefull only if loaded with 'optimize_mem' and the animation also has size optimized frames.

Parameters

    in frame index into frame list

Returns

width in FLTK pixle units
12.9.4.18 frame_x()

int Fl_Anim_GIF_Image::frame_x (int frame) const

Return the frame position of a frame.
Usefull only if loaded with 'optimize_mem' and the animation also has size optimized frames.

Parameters

| in | frame | index into frame list |

Returns

x position in FLTK pixel units

12.9.4.19 frame_y()

int Fl_Anim_GIF_Image::frame_y (int frame) const

Return the frame position of a frame.
Usefull only if loaded with 'optimize_mem' and the animation also has size optimized frames.

Parameters

| in | frame | index into frame list |

Returns

y position in FLTK pixel units

12.9.4.20 frames()

int Fl_Anim_GIF_Image::frames () const

Get the number of frames in the animation.

Returns

the number of frames

12.9.4.21 image() [1/2]

Fl_Image * Fl_Anim_GIF_Image::image () const

Return the current frame image.

Returns

a pointer to the image or NULL if this is not an animation.

12.9.4.22 image() [2/2]

Fl_Image * Fl_Anim_GIF_Image::image (int frame_) const

Return the image of the given frame index.

Parameters

| in | frame | index into list of frames |

Generated by Doxygen
Returns

image data or NULL if the frame number is not valid.

12.9.4.23  is_animated()

bool Fl_Anim_GIF_Image::is_animated ( ) const
Check if this is a valid animation with more than one frame.
The is_animated() method is just a convenience method for testing the valid flag and the frame count being greater 1.

Returns

true if the animation is valid and has multiple frames.

12.9.4.24  load()

bool Fl_Anim_GIF_Image::load ( const char * name,
    const unsigned char * imgdata = NULL,
    size_t imglength = 0 )
Load an animation from a file or from a memory block.
The load() method is either used from the constructor to load the image from the given file, or to re-load an existing animation from another file.

Parameters

| in  | name | path and name of GIF file in the file system, or the image name when reading from memory |
| in  | imgdata | pointer to the start of the GIF image in memory, or NULL to read from a file |
| in  | imglength | length of the GIF image in memory, or 0 |

Returns

true if the animation loaded correctly

12.9.4.25  name()

const char * Fl_Anim_GIF_Image::name ( ) const
Return the name of the played file as specified in the constructor.
If read from a memory block, this returns the name of the animation.

Returns

pointer to a C string

12.9.4.26  next()

bool Fl_Anim_GIF_Image::next ( )
Show the next frame if the animation is stopped.

Returns

true if the animation has frames

12.9.4.27  on_extension_data()

void Fl_Anim_GIF_Image::on_extension_data ( Fl_GIF_Image::GIF_FRAME & f ) [protected], [virtual]
Reimplemented from Fl_GIF_Image.
12.9.4.28 **on_frame_data()**

```cpp
group Fl_Anim_GIF_Image

Fl_Anim_GIF_Image::on_frame_data (Fl_GIF_Image::GIF_FRAME & f) [protected], [virtual]
```

Reimplemented from Fl_GIF_Image.

12.9.4.29 **playing()**

```cpp
bool Fl_Anim_GIF_Image::playing () const [inline]
```

Return if the animation is currently running or stopped.

**Returns**

true if the animation is running

12.9.4.30 **resize()** [1/2]

```cpp
Fl_Anim_GIF_Image & Fl_Anim_GIF_Image::resize (double scale)
```

Resizes the image to the specified size, replacing the current image.

*If DONT_RESIZE_CANVAS is not set, the canvas widget will also be resized.*

**Parameters**

| in  | scale | rescale factor in relation to current size |

12.9.4.31 **resize()** [2/2]

```cpp
Fl_Anim_GIF_Image & Fl_Anim_GIF_Image::resize (int w, int h)
```

Resizes the image to the specified size, replacing the current image.

*If DONT_RESIZE_CANVAS is not set, the canvas widget will also be resized.*

**Parameters**

| in | w,h | new size of the animation frames |

12.9.4.32 **speed()** [1/2]

```cpp
double Fl_Anim_GIF_Image::speed () const
```

Get the animation speed factor.

**Returns**

the current speed factor

12.9.4.33 **speed()** [2/2]

```cpp
void Fl_Anim_GIF_Image::speed (double speed)
```

Set the animation speed factor.

The `speed()` method changes the playing speed to \( \text{speed} \times \text{original speed} \). E.g. to play at half speed call it with 0.5, for double speed with 2.

**Parameters**

| in | speed | floating point speed factor |
12.9.4.34 start()

bool Fl_Anim_GIF_Image::start ()
The start() method (re-)starts the playing of the frames.
Returns
true if the animation has frames

12.9.4.35 stop()

bool Fl_Anim_GIF_Image::stop ()
The stop() method stops the playing of the frames.
Returns
true if the animation has frames

12.9.4.36 uncache()

void Fl_Anim_GIF_Image::uncache () [virtual]
Uncache all cached image data now.
Re-implemented from Fl_Pixmap.
Reimplemented from Fl_Image.

12.9.4.37 valid()

bool Fl_Anim_GIF_Image::valid () const
Check if animation is valid.
Returns
true if the class has successfully loaded and the image has at least one frame.

12.9.5 Member Data Documentation

12.9.5.1 loop

bool Fl_Anim_GIF_Image::loop = true [static]
The loop flag can be used to (dis-)allow loop count.
If set (which is the default), the animation will be stopped after the number of repeats specified in the GIF file
(typically this count is set to 'forever' anyway). If cleared the animation will always be 'forever', regardless of what is
specified in the GIF file.

12.9.5.2 min_delay

double Fl_Anim_GIF_Image::min_delay = 0. [static]
The min_delay value can be used to set a minimum value for the frame delay for playback.
This is to prevent CPU hogs caused by images with very low delay rates. This is a global value for all
Fl_Anim_GIF_Image objects.
The documentation for this class was generated from the following files:
- Fl_Anim_GIF_Image.H
- Fl_Anim_GIF_Image.cxx

12.10 Fl_Bitmap Class Reference

The Fl_Bitmap class supports caching and drawing of mono-color (bitmap) images.
#include <Fl_Bitmap.H>
Inheritance diagram for Fl_Bitmap:
Public Member Functions

- int cache_h ()
- int cache_w ()
- Fl_Image * copy () const
- Fl_Image * copy (int W, int H) const FL_OVERRIDE
  Creates a resized copy of the image.
- void draw (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE
  Draws the image to the current drawing surface with a bounding box.
- Fl_Bitmap (const char * bits, int bits_length, int W, int H)
  The constructors create a new bitmap from the specified bitmap data.
- Fl_Bitmap (const char * bits, int W, int H)
  The constructors create a new bitmap from the specified bitmap data.
- Fl_Bitmap (const uchar * bits, int bits_length, int W, int H)
  The constructors create a new bitmap from the specified bitmap data.
- Fl_Bitmap (const uchar * bits, int W, int H)
  The constructors create a new bitmap from the specified bitmap data.
- void label (Fl_Menu_Item * m) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget * w) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a widget or menu item.
- void uncache () FL_OVERRIDE
  If the image has been cached for display, delete the cache data.
- virtual ~Fl_Bitmap ()
  The destructor frees all memory and server resources that are used by the bitmap.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image * as_shared_image ()
  Returns whether an image is an Fl_Shared_Image or not.
- virtual void color_average (Fl_Color c, float i)
  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * copy () const
  Creates a copy of the image in the same size.
- int count () const
  Returns the number of data values associated with the image.
- int d () const
  Returns the image depth.
- const char *const * data () const
  Returns a pointer to the current image data array.
- int data_h () const
  Returns the height of the image data.
- int data_w () const
  Returns the width of the image data.
Returns the width of the image data.

• virtual void desaturate ()
  The desaturate() method converts an image to grayscale.

• void draw (int X, int Y)
  Draws the image to the current drawing surface.

• int fail () const
  Returns a value that is not 0 if there is currently no image available.

• Fl_Image (int W, int H, int D)
  The constructor creates an empty image with the specified width, height, and depth.

• int h () const
  Returns the current image drawing height in FLTK units.

• void inactive ()
  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

• int ld () const
  Returns the current line data size in bytes.

• virtual void release ()
  Releases an Fl_Image - the same as 'delete this'.

• virtual void scale (int width, int height, int proportional=1, int can_expand=0)
  Sets the drawing size of the image.

• int w () const
  Returns the current image drawing width in FLTK units.

• virtual ~Fl_Image ()
  The destructor is a virtual method that frees all memory used by the image.

Public Attributes

• int alloc_array
  Non-zero if array points to bitmap data allocated internally.

• const uchar * array
  pointer to raw bitmap data

Friends

• class Fl_Graphics_Driver

Additional Inherited Members

Static Public Member Functions inherited from Fl_Image

• static Fl_Labeltype define_FL_IMAGE_LABEL ()
• static Fl_RGB_Scaling RGB_scaling ()
  Returns the currently used RGB image scaling method.
• static void RGB_scaling (Fl_RGB_Scaling)
  Sets the RGB image scaling method used for copy(int, int).
• static Fl_RGB_Scaling scaling_algorithm ()
  Gets what algorithm is used when resizing a source image to draw it.
• static void scaling_algorithm (Fl_RGB_Scaling algorithm)
  Sets what algorithm is used when resizing a source image to draw it.
12.10 Fl_Bitmap Class Reference

Static Public Attributes inherited from Fl_Image

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_MEMORY_ACCESS = -4
- static const int ERR_NO_IMAGE = -1
- static bool register_images_done = false
  
  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_Image

- void d (int D)
  
  Sets the current image depth.
- void data (const char *const *p, int c)
  
  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)
  
  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)
  
  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)
  
  Sets the height of the image data.
- void ld (int LD)
  
  Sets the current line data size in bytes.
- void w (int W)
  
  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void labeltype (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void measure (const Fl_Label *lo, int &lw, int &lh)

12.10.1 Detailed Description

The Fl_Bitmap class supports caching and drawing of mono-color (bitmap) images. Images are drawn using the current color.

12.10.2 Constructor & Destructor Documentation

12.10.2.1 Fl_Bitmap() [1/4]

Fl_Bitmap::Fl_Bitmap ( const uchar * bits, int W, int H ) [inline]

The constructors create a new bitmap from the specified bitmap data.

See also

Fl_Bitmap(const uchar *bits, int bits_length, int W, int H)

12.10.2.2 Fl_Bitmap() [2/4]

Fl_Bitmap::Fl_Bitmap ( const char * bits, int W, int H ) [inline]

The constructors create a new bitmap from the specified bitmap data.
12.10.2.3 Fl_Bitmap()

Fl_Bitmap::Fl_Bitmap (  
    const uchar * bits,  
    int bits_length,    
    int W,              
    int H )

The constructors create a new bitmap from the specified bitmap data. If the provided array is too small to contain all the image data, the constructor will not generate the bitmap to avoid illegal memory read access and instead set data to NULL and ld to ERR_MEMORY_ACCESS.

Parameters

<table>
<thead>
<tr>
<th>bits</th>
<th>bitmap data, one pixel per bit, rows are rounded to the next byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits_length</td>
<td>length of the bits array in bytes</td>
</tr>
<tr>
<td>W</td>
<td>image width in pixels</td>
</tr>
<tr>
<td>H</td>
<td>image height in pixels</td>
</tr>
</tbody>
</table>

See also

Fl_Bitmap(const char *bits, int bits_length, int W, int H), Fl_Bitmap(const uchar *bits, int W, int H)

12.10.2.4 Fl_Bitmap()

Fl_Bitmap::Fl_Bitmap (  
    const char * bits,  
    int bits_length,    
    int W,              
    int H )

The constructors create a new bitmap from the specified bitmap data. If the provided array is too small to contain all the image data, the constructor will not generate the bitmap to avoid illegal memory read access and instead set data to NULL and ld to ERR_MEMORY_ACCESS.

Parameters

<table>
<thead>
<tr>
<th>bits</th>
<th>bitmap data, one pixel per bit, rows are rounded to the next byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits_length</td>
<td>length of the bits array in bytes</td>
</tr>
<tr>
<td>W</td>
<td>image width in pixels</td>
</tr>
<tr>
<td>H</td>
<td>image height in pixels</td>
</tr>
</tbody>
</table>

See also

Fl_Bitmap(const char *bits, int bits_length, int W, int H), Fl_Bitmap(const uchar *bits, int W, int H)

12.10.3 Member Function Documentation

12.10.3.1 copy()

Fl_Image * Fl_Bitmap::copy (  
    int W,  
    int H ) const [virtual]

Creates a resized copy of the image.
The new image should be released when you are done with it.
Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator delete for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's.
The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().
For the new image the following equations are true:

- \( w() = \text{data}_w() = W \)
- \( h() = \text{data}_h() = H \)

Parameters

| in | \( W,H \) | Requested width and height of the new image |

Note

The returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your overridden methods of 'Fl_Image::copy() const' and 'Fl_Image::copy(int, int) const' must also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented from Fl_Image.

12.10.3.2 draw()

```cpp
void Fl_Bitmap::draw (  
    int X,  
    int Y,  
    int W,  
    int H,  
    int cx = 0,  
    int cy = 0 ) [virtual]
```

Draws the image to the current drawing surface with a bounding box.
Arguments \( X,Y,W,H \) specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the \( cx \) and \( cy \) arguments.
In other words: `fl_push_clip(X,Y,W,H)` is applied, the image is drawn with its upper-left corner at \( X-cx,Y-cy \) and its own width and height, `fl_pop_clip()` is applied.
Reimplemented from Fl_Image.

12.10.3.3 label() [1/2]

```cpp
void Fl_Bitmap::label (  
    Fl_Menu_Item * m ) [virtual]
```

This method is an obsolete way to set the image attribute of a menu item.

Deprecated Please use Fl_Menu_Item::image() instead.
Reimplemented from Fl_Image.

12.10.3.4 label() [2/2]

```cpp
void Fl_Bitmap::label (  
    Fl_Widget * widget ) [virtual]
```

This method is an obsolete way to set the image attribute of a widget or menu item.
Deprecated Please use Fl_Widget::image() or Fl_Widget::deimage() instead.

Reimplemented from Fl_Image.

12.10.3.5 uncache()

void Fl_Bitmap::uncache () [virtual]
If the image has been cached for display, delete the cache data.
This allows you to change the data used for the image and then redraw it without recreating an image object.
Reimplemented from Fl_Image.

The documentation for this class was generated from the following files:

- Fl_Bitmap.H
- Fl_Bitmap.cxx

12.11 Fl_BMP_Image Class Reference

The Fl_BMP_Image class supports loading, caching, and drawing of Windows Bitmap (BMP) image files.

#include <Fl_BMP_Image.H>

Inheritance diagram for Fl_BMP_Image:

```
Fl_Image
   Fl_RGB_Image
   Fl_BMP_Image
   Fl_ICO_Image
```

Public Member Functions

- Fl_BMP_Image (const char *filename)
  This constructor loads the named BMP image from the given BMP filename.
- Fl_BMP_Image (const char *imagename, const unsigned char *data, const long length=-1)
  This constructor loads a BMP image from memory.

Public Member Functions inherited from Fl_RGB_Image

- virtual Fl_SVG_Image * as_svg_image ()
  Returns whether an image is an Fl_SVG_Image or not.
- int cache_h ()
- int cache_w ()
- void color_average (FL_Color c, float i) FL_OVERRIDE
  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * copy () const
- Fl_Image * copy (int W, int H) const FL_OVERRIDE
  Creates a resized copy of the image.
- void desaturate () FL_OVERRIDE
  The desaturate() method converts an image to grayscale.
- void draw (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE
  Draws the image to the current drawing surface with a bounding box.
• **Fl_RGB_Image (const Fl_Pixmap *pxm, Fl_Color bg=FL_GRAY)**
  The constructor creates a new RGBA image from the specified Fl_Pixmap.

• **Fl_RGB_Image (const uchar *bits, int bits_length, int W, int H, int D, int LD)**
  The constructor creates a new image from the specified data.

• **Fl_RGB_Image (const uchar *bits, int W, int H, int D=3, int LD=0)**
  The constructor creates a new image from the specified data.

• **void label (Fl_Menu_Item *m) FL_OVERRIDE**
  This method is an obsolete way to set the image attribute of a menu item.

• **void label (Fl_Widget *w) FL_OVERRIDE**
  This method is an obsolete way to set the image attribute of a widget or menu item.

• **virtual void normalize ()**
  Makes sure the object is fully initialized.

• **void uncache () FL_OVERRIDE**
  If the image has been cached for display, delete the cache data.

• **~Fl_RGB_Image () FL_OVERRIDE**
  The destructor frees all memory and server resources that are used by the image.

### Public Member Functions inherited from Fl_Image

• **virtual class Fl_Shared_Image * as_shared_image ()**
  Returns whether an image is an Fl_Shared_Image or not.

• **Fl_Image * copy () const**
  Creates a copy of the image in the same size.

• **int count () const**
  Returns the number of data values associated with the image.

• **int d () const**
  Returns the image depth.

• **const char * const * data () const**
  Returns a pointer to the current image data array.

• **int data_h () const**
  Returns the height of the image data.

• **int data_w () const**
  Returns the width of the image data.

• **void draw (int X, int Y)**
  Draws the image to the current drawing surface.

• **int fail () const**
  Returns a value that is not 0 if there is currently no image available.

• **Fl_Image (int W, int H, int D)**
  The constructor creates an empty image with the specified width, height, and depth.

• **int h () const**
  Returns the current image drawing height in FLTK units.

• **virtual void release ()**
  Releases an Fl_Image - the same as ‘delete this’.

• **virtual void scale (int width, int height, int proportional=1, int can_expand=0)**
  Sets the drawing size of the image.

• **int w () const**
  Returns the current image drawing width in FLTK units.

• **virtual ~Fl_Image ()**
  The destructor is a virtual method that frees all memory used by the image.
Protected Member Functions

• void load_bmp_(class Fl_Image_Reader &rdr, int ico_height=0, int ico_width=0)

Protected Member Functions inherited from Fl_Image

• void d (int D)
  Sets the current image depth.
• void data (const char * const *p, int c)
  Sets the current data pointer and count of pointers in the array.
• void draw_empty (int X, int Y)
  The protected method draw_empty() draws a box with an X in it.
• int draw_scaled (int X, int Y, int W, int H)
  Draw the image to the current drawing surface rescaled to a given width and height.
• void h (int H)
  Sets the height of the image data.
• void ld (int LD)
  Sets the current line data size in bytes.
• void w (int W)
  Sets the width of the image data.

Additional Inherited Members

Static Public Member Functions inherited from Fl_RGB_Image

• static size_t max_size ()
  Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
• static void max_size (size_t size)
  Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

Static Public Member Functions inherited from Fl_Image

• static Fl_Labeltype define_FL_IMAGE_LABEL ()
• static Fl_RGB_Scaling RGB_scaling ()
  Returns the currently used RGB image scaling method.
• static void RGB_scaling (Fl_RGB_Scaling)
  Sets the RGB image scaling method used for copy(int, int).
• static Fl_RGB_Scaling scaling_algorithm ()
  Gets what algorithm is used when resizing a source image to draw it.
• static void scaling_algorithm (Fl_RGB_Scaling algorithm)
  Sets what algorithm is used when resizing a source image to draw it.

Public Attributes inherited from Fl_RGB_Image

• int alloc_array
  If non-zero, the object's data array is delete[]d when deleting the object.
• const uchar * array
  Points to the start of the object's data array.
**Static Public Attributes inherited from Fl_Image**

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_MEMORY_ACCESS = -4
- static const int ERR_NO_IMAGE = -1
- static bool register_images_done = false

  *True after fl_register_images() was called, false before.*

**Static Protected Member Functions inherited from Fl_Image**

- static void labeltype (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void measure (const Fl_Label *lo, int &lw, int &lh)

### 12.11 Detailed Description

The Fl_BMP_Image class supports loading, caching, and drawing of Windows Bitmap (BMP) image files.

### 12.11.2 Constructor & Destructor Documentation

#### 12.11.2.1 Fl_BMP_Image() [1/2]

Fl_BMP_Image::Fl_BMP_Image (const char * filename)

This constructor loads the named BMP image from the given BMP filename. The destructor frees all memory and server resources that are used by the image.

Use Fl_Image::fail() to check if Fl_BMP_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the BMP format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

**Parameters**

- **in** filename a full path and name pointing to a BMP file.

#### 12.11.2.2 Fl_BMP_Image() [2/2]

Fl_BMP_Image::Fl_BMP_Image (const char * imagename, const unsigned char * data, const long length = -1)

This constructor loads a BMP image from memory.

Construct an image from a block of memory inside the application. Fluid offers "binary data" chunks as a great way to add image data into the C++ source code. imagename can be NULL. If a name is given, the image is added to the list of shared images and will be available by that name.

The destructor frees all memory and server resources that are used by the image.

The (new and optional) third parameter length should be used so buffer overruns (i.e. truncated images) can be checked. See note below.

If length is not used

- it defaults to -1 (unlimited size)
- buffer overruns will not be checked.

See also

Fl_BMP_Image::Fl_BMP_Image(const char* imagename, const unsigned char* data, const long length = -1);
The optional parameter `length` is available since FLTK 1.4.0. Not using it is deprecated and old code should be modified to use it. This parameter will likely become mandatory in a future FLTK version.

Use `Fl_Image::fail()` to check if `Fl_BMP_Image` failed to load. `fail()` returns `ERR_FILE_ACCESS` if the image could not be read from memory, `ERR_FORMAT` if the BMP format could not be decoded, and `ERR_NO_IMAGE` if the image could not be loaded for another reason.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>imagename</code></td>
<td>A name given to this image or NULL</td>
</tr>
<tr>
<td><code>data</code></td>
<td>Pointer to the start of the BMP image in memory.</td>
</tr>
<tr>
<td><code>length</code></td>
<td>Length of the BMP image in memory.</td>
</tr>
</tbody>
</table>

See also

- `Fl_BMP_Image::Fl_BMP_Image(const char *filename)`
- `Fl_Shared_Image`

The documentation for this class was generated from the following files:

- `Fl_BMP_Image.h`
- `Fl_BMP_Image.cxx`

## 12.12 Fl_Box Class Reference

This widget simply draws its box, and possibly its label.

```c
#include <Fl_Box.h>
```

Inheritance diagram for `Fl_Box`:

```
Fl_Box
|___ Fl_Widget
```

### Public Member Functions

- **`Fl_Box`** ([`Fl_Boxtype` b, int X, int Y, int W, int H, const char *l])
  
  See `Fl_Box::Fl_Box(int x, int y, int w, int h, const char * = 0)`

- **`Fl_Box`** (int X, int Y, int W, int H, const char *l=0)

- **int handle (int)** FL_OVERRIDE

  Handles the specified event.

### Public Member Functions inherited from Fl_Widget

- void `_clear_fullscreen` ()
- void `_set_fullscreen` ()
- void activate ()

  Activates the widget.

- unsigned int active () const

  Returns whether the widget is active.

- int active_r () const

  Returns whether the widget and all of its parents are active.
• Fl_Align align () const
  Gets the label alignment.

• void align (Fl_Align alignment)
  Sets the label alignment.

• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.

• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const ∗ as_gl_window () const

• virtual Fl_Group ∗ as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Group const ∗ as_group () const

• virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window const ∗ as_window () const

• void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.

• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

• void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.

• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

• Fl_Boxtype box () const
  Gets the box type of the widget.

• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.

• Fl_Callback_p callback () const
  Gets the current callback function for the widget.

• void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

• void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.

• void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.

• unsigned int changed () const
  Checks if the widget value changed since the last callback.

• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

• void clear_changed ()
  Marks the value of the widget as unchanged.

• void clear_damage (uchar c=0)
  Clears or sets the damage flags.

• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.
• void copy_label (const char *new_label)
  Sets the current label.
• void copy_tooltip (const char *text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reas  on reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
Makes a widget invisible.

- `FL_Image * image ()`
  Gets the image that is used as part of the widget label when in the active state.

- `const FL_Image * image () const`
  Gets the image that is used as part of the widget label when in the active state.

- `void image (FL_Image &img)`
  Sets the image to use as part of the widget label when in the active state.

- `void image (FL_Image * img)`
  Sets the image to use as part of the widget label when in the active state.

- `int image_bound () const`
  Returns whether the image is managed by the widget.

- `int inside (const Fl_Widget * wgt) const`
  Checks if this widget is a child of `wgt`.

- `int is_label_copied () const`
  Returns whether the current label was assigned with `copy_label()`.

- `const char * label () const`
  Gets the current label text.

- `void label (const char * text)`
  Sets the current label pointer.

- `void label (FL_Labeltype a, const char * b)`
  Shortcut to set the label text and type in one call.

- `FL_Color labelcolor () const`
  Gets the label color.

- `void labelcolor (FL_Color c)`
  Sets the label color.

- `FL_Font labelfont () const`
  Gets the font to use.

- `void labelfont (FL_Font f)`
  Sets the font to use.

- `FL_Fontsize labelsize () const`
  Gets the font size in pixels.

- `void labelsize (FL_Fontsize pix)`
  Sets the font size in pixels.

- `FL_Labeltype labeltype () const`
  Gets the label type.

- `void labeltype (FL_Labeltype a)`
  Sets the label type.

- `void measure_label (int &ww, int &hh) const`
  Sets width `ww` and height `hh` accordingly with the label size.

- `bool needs_keyboard () const`
  Returns whether this widget needs a keyboard.

- `void needs_keyboard (bool needs)`
  Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  Returns if a widget is used for output only.

- `FL_Group * parent () const`
  Returns a pointer to the parent widget.

- `void parent (FL_Group * p)`
  Internal use only - "for hacks only".

- `void position (int X, int Y)`
  Repositions the window or widget.
• void **redraw** ()
  Schedules the drawing of the widget.

• void **redraw_label** ()
  Schedules the drawing of the label.

• virtual void **resize** (int x, int y, int w, int h)
  Changes the size or position of the widget.

• Fl_Color **selection_color** () const
  Gets the selection color.

• void **selection_color** (Fl_Color a)
  Sets the selection color.

• void **set_active** ()
  Marks the widget as active without sending events or changing focus.

• void **set_changed** ()
  Marks the value of the widget as changed.

• void **set_output** ()
  Sets a widget to output only.

• void **set_visible** ()
  Makes the widget visible.

• void **set_visible_focus** ()
  Enables keyboard focus navigation with this widget.

• int **shortcut_label** () const
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.

• void **shortcut_label** (int value)
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.

• virtual void **show** ()
  Makes a widget visible.

• void **size** (int W, int H)
  Changes the size of the widget.

• int **take_focus** ()
  Gives the widget the keyboard focus.

• unsigned int **takeevents** () const
  Returns if the widget is able to take events.

• int **test_shortcut** ()
  Returns true if the widget’s label contains the entered ‘&x’ shortcut.

• const char ∗ **tooltip** () const
  Gets the current tooltip text.

• void **tooltip** (const char ∗text)
  Sets the current tooltip text.

• Fl_Window ∗ **top_window** () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window ∗ **top_window_offset** (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar **type** () const
  Gets the widget type.

• void **type** (uchar t)
  Sets the widget type.

• int **use_accents_menu** ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void ∗ **user_data** () const
  Gets the user data for this widget.

• void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)
Sets the user data for this widget.

- **void** `user_data (void ∗v)`
  Sets the user data for this widget.

- **unsigned int** `visible () const`
  Returns whether a widget is visible.

- **unsigned int** `visible_focus () const`
  Checks whether this widget has a visible focus.

- **void** `visible_focus (int v)`
  Modifies keyboard focus navigation.

- **int** `visible_r () const`
  Returns whether a widget and all its parents are visible.

- **int** `w () const`
  Gets the widget width.

- **Fl_When** `when () const`
  Returns the conditions under which the callback is called.

- **void** `when (uchar i)`
  Sets the flags used to decide when a callback is called.

- **Fl_Window ∗** `window () const`
  Returns a pointer to the nearest parent window up the widget hierarchy.

- **int** `x () const`
  Gets the widget position in its window.

- **int** `y () const`
  Gets the widget position in its window.

- **virtual** `~Fl_Widget ()`
  Destroys the widget.

Protected Member Functions

- **void** `draw () FL_OVERRIDE`
  Draws the widget.

Protected Member Functions inherited from Fl_Widget

- **void** `clear_flag (unsigned int c)`
  Clears a flag in the flags mask.

- **void** `draw_backdrop () const`
  If `FL_ALIGN_IMAGE_BACKDROP` is set, the image or deimage will be drawn.

- **void** `draw_box () const`
  Draws the widget box according its box style.

- **void** `draw_box (Fl_Boxtype t, Fl_Color c) const`
  Draws a box of type `t`, of color `c` at the widget's position and size.

- **void** `draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const`
  Draws a box of type `t`, of color `c` at the position `X,Y` and size `W,H`.

- **void** `draw_focus () const`
  Draws a focus rectangle around the widget.

- **void** `draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const`
  Draws a focus box for the widget at the given position and size.

- **void** `draw_label () const`
  Draws the widget's label at the defined label position.

- **void** `draw_label (int, int, int, int) const`
Draws the label in an arbitrary bounding box.

- **Fl_Widget**(int x, int y, int w, int h, const char ∗label=0L)
  
  Creates a widget at the given position and size.

- unsigned int **flags**() const
  
  Gets the widget flags mask.

- void **h**(int v)
  
  Internal use only.

- void **set_flag**(unsigned int c)
  
  Sets a flag in the flags mask.

- void **w**(int v)
  
  Internal use only.

- void **x**(int v)
  
  Internal use only.

- void **y**(int v)
  
  Internal use only.

**Additional Inherited Members**

**Static Public Member Functions inherited from Fl_Widget**

- static void **default_callback**(Fl_Widget ∗widget, void ∗data)
  
  The default callback for all widgets that don’t set a callback.

- static unsigned int **label_shortcut**(const char ∗t)
  
  Returns the Unicode value of the ‘&x’ shortcut in a given text.

- static int **test_shortcut**(const char ∗t, const bool require_alt=false)
  
  Returns true if the given text t contains the entered ‘&x’ shortcut.

**Protected Types inherited from Fl_Widget**

- enum {
    INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
    FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
    OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
    MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
    GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
    NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22, AUTO_DELETE_USER_DATA = 1<<23,
    MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31}

  flags possible values enumeration.

**12.12.1 Detailed Description**

This widget simply draws its box, and possibly its label. Putting it before some other widgets and making it big enough to surround them will let you draw a frame around them.
12.12.2 Constructor & Destructor Documentation

12.12.2.1 Fl_Box()

Fl_Box::Fl_Box (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char ∗ l = 0 )

- The first constructor sets box() to FL_NO_BOX, which means it is invisible. However such widgets are useful as placeholders or Fl_Group::resizable() values. To change the box to something visible, use box(n).

- The second form of the constructor sets the box to the specified box type.

The destructor removes the box.

12.12.3 Member Function Documentation

12.12.3.1 draw()

void Fl_Box::draw ( ) [protected], [virtual]

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

12.12.3.2 handle()

int Fl_Box::handle (  
    int event ) [virtual]

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class.

This requires knowledge of the details of the inherited class.

In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in | event | the kind of event received |

Return values

| 0 | if the event was not used or understood |
| 1 | if the event was used and can be deleted |

Generated by Doxygen
See also

Fl_Event

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

- Fl_Box.H
- Fl_Box.cxx

12.13 Fl_Browser Class Reference

The Fl_Browser widget displays a scrolling list of text lines, and manages all the storage for the text.

#include <Fl_Browser.H>

Inheritance diagram for Fl_Browser:

```
Fl_Widget
    |
    v
Fl_Group
    |
    v
Fl_Browser_
    |
    v
Fl_Browser
    |
Fl_File_Browser
    |
Fl_Hold_Browser
    |
Fl_Multi_Browser
    |
Fl_Select_Browser
```

Public Types

- enum Fl_Line_Position { TOP, BOTTOM, MIDDLE }

  For internal use only?

Public Types inherited from Fl_Browser_

- enum{
  HORIZONTAL = 1, VERTICAL = 2, BOTH = 3, ALWAYS_ON = 4,
  HORIZONTAL_ALWAYS = 5, VERTICAL_ALWAYS = 6, BOTH_ALWAYS = 7 }

  Values for has_scrollbar().

Public Member Functions

- void add (const char *newtext, void *d=0)
  Adds a new line to the end of the browser.
- void bottomline (int line)
  Scrolls the browser so the bottom item in the browser is showing the specified line.
- void clear ()
  Removes all the lines in the browser.
- char column_char () const
  Gets the current column separator character.
- void column_char (char c)
  Sets the column separator to c.
- const int * column_widths () const
  Gets the current column width array.
- void column_widths (const int *arr)
Sets the current array to `arr`.

- **void** `∗` **data** (int line) **const**
  
  Returns the user data() for specified line.

- **void** **data** (int line, void `∗`**d**)
  
  Sets the user data for specified line to `d`.

- **void** **display** (int line, int val=1)
  
  For back compatibility.

- **int** **displayed** (int line) **const**
  
  Returns non-zero if line has been scrolled to a position where it is being displayed.

- **Fl_Browser** (int X, int Y, int W, int H, const char `∗`**L=0**)
  
  The constructor makes an empty browser.

- **char** **format_char** () **const**
  
  Gets the current format code prefix character, which by default is '@'.

- **void** **format_char** (char c)
  
  Sets the current format code prefix character to `c`.

- **void** **hide** () **FL_OVERRIDE**
  
  Hides the entire Fl_Browser widget – opposite of show().

- **void** **hide** (int line)
  
  Makes line invisible, preventing selection by the user.

- **Fl_Image ∗** **icon** (int line) **const**
  
  Returns the icon currently defined for line.

- **void** **icon** (int line, Fl_Image `∗`**icon**)
  
  Set the image icon for line to the value `icon`.

- **void** **insert** (int line, const char `∗`**newtext**, void `∗`**d=0**)
  
  Insert a new entry whose label is `newtext` above given line, optional data `d`.

- **void** **lineposition** (int line, Fl_Line_Position pos)
  
  Updates the browser so that line is shown at position `pos`.

- **int** **load** (const char `∗`**filename**)
  
  Clears the browser and reads the file, adding each line from the file to the browser.

- **void** **make_visible** (int line)
  
  Make the item at the specified line visible().

- **void** **middleline** (int line)
  
  Scrolls the browser so the middle item in the browser is showing the specified line.

- **void** **move** (int to, int from)
  
  Line from is removed and reinserted at to.

- **void** **remove** (int line)
  
  Remove entry for given line number, making the browser one line shorter.

- **void** **remove_icon** (int line)
  
  Removes the icon for line.

- **void** **replace** (int a, const char `∗`**b**)
  
  For back compatibility only.

- **int** **select** (int line, int val=1)
  
  Sets the selection state of the item at line to the value `val`.

- **int** **selected** (int line) **const**
  
  Returns 1 if specified line is selected, 0 if not.

- **void** **show** () **FL_OVERRIDE**
  
  Shows the entire Fl_Browser widget – opposite of hide().

- **void** **show** (int line)
  
  Makes line visible, and available for selection by user.

- **int** **size** () **const**
  
  Returns how many lines are in the browser.
• void **size**(int W, int H)

  Swaps two browser lines a and b.

• const char * **text**(int line) const

  Returns the label text for the specified line.

• void **swap**(int a, int b)

  Swaps two browser lines a and b.

• const char * **text**(int line, const char *newtext)

  Sets the text for the specified line to newtext.

• **Fl_Fontsize** **textsize**() const

  Gets the default text size (in pixels) for the lines in the browser.

• void textsize (**Fl_Fontsize** newSize)

  Sets the default text size (in pixels) for the lines in the browser to newSize.

• int **topline**(int line) const

  Returns the line that is currently visible at the top of the browser.

• void topline (int line)

  Scrolls the browser so the top item in the browser is showing the specified line.

• int **value**(int line) const

  Returns the line number of the currently selected line, or 0 if none selected.

• void value (int line)

  Sets the browser’s value(), which selects the specified line.

• int **visible**(int line) const

  Returns non-zero if the specified line is visible, 0 if hidden.

• ~**Fl_Browser**()

  The destructor deletes all list items and destroys the browser.

**Public Member Functions inherited from Fl_Browser**

• int deselect (int docallbacks=0)

  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.

• void display (void *item)

  Displays the item, scrolling the list as necessary.

• int handle (int event) FL_OVERRIDE

  Handles the event within the normal widget bounding box.

• uchar has_scrollbar () const

  Returns the current scrollbar mode, see Fl_Browser::has_scrollbar(uchar)

• void has_scrollbar (uchar mode)

  Sets whether the widget should have scrollbars or not (default Fl_Browser::BOTH).

• int hposition () const

  Gets the horizontal scroll position of the list as a pixel position pos.

• void hposition (int)

  Sets the horizontal scroll position of the list to pixel position pos.

• int linespacing () const

  Return the height of additional spacing between browser lines.

• void linespacing (int pixels)

  Add some space between browser lines.

• int position () const

• void position (int pos)

• void position (int x, int y)

• void resize (int X, int Y, int W, int H) FL_OVERRIDE

  Repositions and/or resizes the browser.

• void scrollbar_left ()

  Moves the vertical scrollbar to the lefthand side of the list.
• void scrollbar_right ()
  Moves the vertical scrollbar to the right hand side of the list.

• int scrollbar_size () const
  Gets the current size of the scrollbars' troughs, in pixels.

• void scrollbar_size (int newSize)
  Sets the pixel size of the scrollbars' troughs to newSize, in pixels.

• int scrollbar_width () const
  Returns the global value Fl::scrollbar_size().

• void scrollbar_width (int width)
  Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

• int select (void ∗ item, int val=1, int docallbacks=0)
  Sets the selection state of item to val, and returns 1 if the state changed or 0 if it did not.

• int select_only (void ∗ item, int docallbacks=0)
  Selects item and returns 1 if the state changed or 0 if it did not.

• void sort (int flags=0)
  Sort the items in the browser based on flags.

• Fl_Color textcolor () const
  Gets the default text color for the lines in the browser.

• void textcolor (Fl_Color col)
  Sets the default text color for the lines in the browser to color col.

• Fl_Font textfont () const
  Gets the default text font for the lines in the browser.

• void textfont (Fl_Font font)
  Sets the default text font for the lines in the browser to font.

• Fl_Fontsize textsize () const
  Gets the default text size (in pixels) for the lines in the browser.

• void textsize (Fl_Fontsize newSize)
  Sets the default text size (in pixels) for the lines in the browser to size.

• int vposition () const
  Gets the vertical scroll position of the list as a pixel position pos.

• void vposition (int pos)
  Sets the vertical scroll position of the list to pixel position pos.

Public Member Functions inherited from Fl_Group

• Fl_Widget ∗ & _ddfdesign_kludge ()
  This is for forms compatibility only.

• void add (Fl_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.

• void add (Fl_Widget ∗&)
  See void Fl_Group::add(Fl_Widget &w)

• void addResizable (Fl_Widget &)
  Adds a widget to the group and makes it the resizable widget.

• Fl_Widget ∗ & & _ddfdesign_kludge ()
  This is for forms compatibility only.

• void add (Fl_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.

• void add (Fl_Widget ∗&)
  See void Fl_Group::add(Fl_Widget &w)

• void addResizable (Fl_Widget &)
  Adds a widget to the group and makes it the resizable widget.

• Fl_Widget ∗ const + array () const
  Returns a pointer to the array of children.

• Fl_Group ∗ & as_group () const FL_OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• Fl_Group ∗ & as_group () FL_OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• void begin ()
  Sets the current group so you can build the widget tree by just constructing the widgets.

• Fl_Widget ∗ child (int n) const
Returns array()[n].  

- **int **`children`() const  
  Returns how many child widgets the group has.  
- **void **`clear`()  
  Deletes all child widgets from memory recursively.  
- **unsigned int** `clip_children`()  
  Returns the current clipping mode.  
- **void **`clip_children`(int c)  
  Controls whether the group widget clips the drawing of child widgets to its bounding box.  
- **virtual int **`delete_child`(int n)  
  Removes the widget at index from the group and deletes it.  
- **void **`end`()  
  Exactly the same as current((this->parent()).  
- **int **`find`(const Fl_Widget &o) const  
  See int Fl_Group::find(const Fl_Widget *w) const.  
- **int **`find`(const Fl_Widget *) const  
  Searches the child array for the widget and returns the index.  
- **Fl_Group** (int, int, int, int, const char *=0)  
  Creates a new Fl_Group widget using the given position, size, and label string.  
- **void **`focus`(Fl_Widget *W)  
- **void **`forms_end`()  
  This is for forms compatibility only.  
- **int **`handle`(int) FL_OVERRIDE  
  Handles the specified event.  
- **void **`init_sizes`()  
  Resets the internal array of widget sizes and positions.  
- **void **`insert`(Fl_Widget &, int i)  
  The widget is removed from its current group (if any) and then inserted into this group.  
- **void **`insert`(Fl_Widget &, Fl_Widget *before)  
  This does insert(w, find(before)).  
- **void **`remove`(Fl_Widget &)  
  Removes a widget from the group but does not delete it.  
- **void **`remove`(Fl_Widget *o)  
  Removes the widget o from the group.  
- **void **`remove`(int index)  
  Removes the widget at index from the group but does not delete it.  
- **Fl_Widget **`resizable`() const  
  Returns the group’s resizable widget.  
- **void **`resizable`(Fl_Widget &o)  
  Sets the group’s resizable widget.  
- **void **`resizable`(Fl_Widget *)  
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.  
- **void **`resize`(int, int, int, int) FL_OVERRIDE  
  Resizes the Fl_Group widget and all of its children.  
- **virtual ~Fl_Group()**  
  The destructor also deletes all the children.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ as_gl_window () const
- virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ as_window () const
- void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
  Checks if the widget value changed since the last callback.
- void clear_active ()

Generated by Doxygen
Marks the widget as inactive without sending events or changing focus.

- **void clear_changed ()**
  Marks the value of the widget as unchanged.

- **void clear_damage (uchar c=0)**
  Clears or sets the damage flags.

- **void clear_output ()**
  Sets a widget to accept input.

- **void clear_visible ()**
  Hides the widget.

- **void clear_visible_focus ()**
  Disables keyboard focus navigation with this widget.

- **Fl_Color color () const**
  Gets the background color of the widget.

- **void color (Fl_Color bg)**
  Sets the background color of the widget.

- **void color (Fl_Color bg, Fl_Color sel)**
  Sets the background and selection color of the widget.

- **Fl_Color color2 () const**
  For back compatibility only.

- **void color2 (unsigned a)**
  For back compatibility only.

- **int contains (const Fl_Widget *w) const**
  Checks if w is a child of this widget.

- **void copy_label (const char *new_label)**
  Sets the current label.

- **void copy_tooltip (const char *text)**
  Sets the current tooltip text.

- **uchar damage () const**
  Returns non-zero if draw() needs to be called.

- **void damage (uchar c)**
  Sets the damage bits for the widget.

- **void damage (uchar c, int x, int y, int w, int h)**
  Sets the damage bits for an area inside the widget.

- **int damage_resize (int, int, int, int)**
  Internal use only.

- **void deactivate ()**
  Deactivates the widget.

- **Fl_Image * deimage ()**
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image * deimage () const**
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (Fl_Image &img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const**
  Returns whether the inactive image is managed by the widget.

- **void do_callback (FL_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with default arguments.

- **void do_callback (Fl_Widget *widget, long arg, FL_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget ∗widget, void ∗arg=0, FL_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• Fl_Image ∗image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image ∗image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget ∗wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char ∗label () const
  Gets the current label text.
• void label (const char ∗text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char ∗b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &,int &,int &,int &)
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group ∗parent () const
Returns a pointer to the parent widget.

- **void** parent (*Fl_Group* *p*)
  
  Internal use only - "for hacks only".

- **void** position (int X, int Y)
  
  Repositions the window or widget.

- **void** redraw ()
  
  Schedules the drawing of the widget.

- **void** redraw_label ()
  
  Schedules the drawing of the label.

- **Fl_Color** selection_color () const
  
  Gets the selection color.

- **void** selection_color (*Fl_Color* a)
  
  Sets the selection color.

- **void** set_active ()
  
  Marks the widget as active without sending events or changing focus.

- **void** set_changed ()
  
  Marks the value of the widget as changed.

- **void** set_output ()
  
  Sets a widget to output only.

- **void** set_visible ()
  
  Makes the widget visible.

- **void** set_visible_focus ()
  
  Enables keyboard focus navigation with this widget.

- **int** shortcut_label () const
  
  Returns whether the widget's label uses '&' to indicate shortcuts.

- **void** shortcut_label (int value)
  
  Sets whether the widget's label uses '&' to indicate shortcuts.

- **void** size (int W, int H)
  
  Changes the size of the widget.

- **int** take_focus ()
  
  Gives the widget the keyboard focus.

- **unsigned int** takesevents () const
  
  Returns if the widget is able to take events.

- **int** test_shortcut ()
  
  Returns true if the widget's label contains the entered 'x' shortcut.

- **const char** * tooltip () const
  
  Gets the current tooltip text.

- **void** tooltip (const char *text)
  
  Sets the current tooltip text.

- **Fl_Window** * top_window () const
  
  Returns a pointer to the top-level window for the widget.

- **Fl_Window** * top_window_offset (int &xoff, int &yoff) const
  
  Finds the x/y offset of the current widget relative to the top-level window.

- **uchar** type () const
  
  Gets the widget type.

- **void** type (uchar t)
  
  Sets the widget type.

- **int** use_accents_menu ()
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- **void** * user_data () const
  
  Gets the user data for this widget.
• void **user_data** (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void **user_data** (void *v)
  Sets the user data for this widget.
• unsigned int **visible** () const
  Returns whether a widget is visible.
• unsigned int **visible_focus** () const
  Checks whether this widget has a visible focus.
• void **visible_focus** (int v)
  Modifies keyboard focus navigation.
• int **visible_r** () const
  Returns whether a widget and all its parents are visible.
• int **w** () const
  Gets the widget width.
• Fl_When **when** () const
  Returns the conditions under which the callback is called.
• void **when** (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * **window** () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int **x** () const
  Gets the widget position in its window.
• int **y** () const
  Gets the widget position in its window.
• virtual ~**Fl_Widget** ()
  Destroys the widget.

Protected Member Functions

• FL_BLINE * _remove (int line)
  Removes the item at the specified line.
• FL_BLINE * **find_line** (int line) const
  Returns the item for specified line.
• int **full_height** () const FL_OVERRIDE
  The height of the entire list of all visible() items in pixels.
• int **incr_height** () const FL_OVERRIDE
  The default ‘average’ item height (including inter-item spacing) in pixels.
• void **insert** (int line, FL_BLINE * item)
  Insert specified item above line.
• void * **item_at** (int line) const FL_OVERRIDE
  Return the item at specified line.
• void **item_draw** (void *item, int X, int Y, int W, int H) const FL_OVERRIDE
  Draws item at the position specified by XYWH.
• void **item_first** () const FL_OVERRIDE
  Returns the very first item in the list.
• int **item_height** (void *item) const FL_OVERRIDE
  Returns height of item in pixels.
• void **item_last** () const FL_OVERRIDE
  Returns the very last item in the list.
• void **item_next** (void *item) const FL_OVERRIDE
  Returns the next item after item.
• void *item_prev (void *item) const FL_OVERRIDE
  Returns the previous item before item.
• void item_select (void *item, int val) FL_OVERRIDE
  Change the selection state of item to the value val.
• int item_selected (void *item) const FL_OVERRIDE
  See if item is selected.
• void item_swap (void *a, void *b) FL_OVERRIDE
  Swap the items a and b.
• const char *item_text (void *item) const FL_OVERRIDE
  Returns the label text for item.
• int item_width (void *item) const FL_OVERRIDE
  Returns width of item in pixels.
• int lineno (void *item) const
  Returns line number corresponding to item, or zero if not found.
• void swap (FL_BLINE *a, FL_BLINE *b)
  Swap the two items a and b.

Protected Member Functions inherited from Fl_Browser_

• void bbox (int &X, int &Y, int &W, int &H) const
  Returns the bounding box for the interior of the list's display window, inside the scrollbars.
• void deleting (void *item)
  This method should be used when item is being deleted from the list.
• int displayed (void *item) const
  Returns non-zero if item has been scrolled to a position where it is being displayed.
• void draw () FL_OVERRIDE
  Draws the list within the normal widget bounding box.
• void *find_item (int ypos)
  This method returns the item under mouse y position ypos.
• Fl_Browser_ (int X, int Y, int W, int H, const char *L=0)
  The constructor makes an empty browser.
• virtual int full_width () const
  This method may be provided by the subclass to indicate the full width of the item list, in pixels.
• void inserting (void *a, void *b)
  This method should be used when an item is in the process of being inserted into the list.
• virtual int item_quick_height (void *item) const
  This method may be provided by the subclass to return the height of the item, in pixels.
• int leftedge () const
  This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.
• void new_list ()
  This method should be called when the list data is completely replaced or cleared.
• void redraw_line (void *item)
  This method should be called when the contents of item has changed, but not its height.
• void redraw_lines ()
  This method will cause the entire list to be redrawn.
• void replacing (void *a, void *b)
  This method should be used when item a is being replaced by item b.
• void *selection () const
  Returns the item currently selected, or NULL if there is no selection.
• void swapping (void *a, void *b)
  This method should be used when two items a and b are being swapped.
• void *top () const
  Returns the item that appears at the top of the list.
Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
- void draw () FL_OVERRIDE
  Draws the widget.
- void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.
- void draw_children ()
  Draws all children of the group.
- void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)
  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()
  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y , int W, int H) const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void draw_label () const
  Draws the widget's label at the defined label position.
- void draw_label (int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.
- unsigned int flags () const
  Gets the widget flags mask.
- void h (int v)
  Internal use only.
- void set_flag (unsigned int c)
Sets a flag in the flags mask.
- void w (int v)
  Internal use only.
- void x (int v)
  Internal use only.
- void y (int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group
- static Fl_Group * current ()
  Returns the currently active group.
- static void current (Fl_Group *g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget
- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don’t set a callback.
- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the ‘&x’ shortcut in a given text.
- static int test_shortcut (const char *t, const bool require_alt=false)
  Returns true if the given text t contains the entered ‘&x’ shortcut.

Public Attributes inherited from Fl_Browser_
- Fl_Scrollbar hscrollbar
  Horizontal scrollbar.
- Fl_Scrollbar scrollbar
  Vertical scrollbar.

Protected Types inherited from Fl_Widget
- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.
12.13.1 Detailed Description

The Fl_Browser widget displays a scrolling list of text lines, and manages all the storage for the text. This is not a text editor or spreadsheet! But it is useful for showing a vertical list of named objects to the user.

![Figure 12.2 Fl_Hold_Browser](image)

Each line in the browser is identified by number. The numbers start at one (this is so that zero can be reserved for “no line” in the selective browsers). Unless otherwise noted, the methods do not check to see if the passed line number is in range and legal. It must always be greater than zero and \(< size()\).

Each line contains a null-terminated string of text and a void * data pointer. The text string is displayed, the void * pointer can be used by the callbacks to reference the object the text describes.

The base class does nothing when the user clicks on it. The subclasses Fl_Select_Browser, Fl_Hold_Browser, and Fl_Multi_Browser react to user clicks to select lines in the browser and do callbacks.

The base class Fl_Browser provides the scrolling and selection mechanisms of this and all the subclasses, but the dimensions and appearance of each item are determined by the subclass. You can use Fl_Browser to display information other than text, or text that is dynamically produced from your own data structures. If you find that loading the browser is a lot of work or is inefficient, you may want to make a subclass of Fl_Browser.

Some common coding patterns used for working with Fl_Browser:

```c
// How to loop through all the items in the browser
for ( int t=1; t<browser->size(); t++ ) { // index 1 based..!
    printf("item %d, label="%s\n", t, browser->text(t));
}
```

Note: If you are subclassing Fl_Browser, it’s more efficient to use the protected methods item_first() and item_next(), since Fl_Browser internally uses linked lists to manage the browser’s items. For more info, see find_item(int).

12.13.2 Constructor & Destructor Documentation

12.13.2.1 Fl_Browser()

Fl_Browser::Fl_Browser {
    int X,
    int Y,
    int W,
    int H,
    const char * L = 0)

The constructor makes an empty browser.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X,Y,W,H</th>
<th>position and size.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>label string, may be NULL.</td>
</tr>
</tbody>
</table>
12.13.3  Member Function Documentation

12.13.3.1  _remove()

```c
FL_BLINE * Fl_Browser::_remove ( int line ) [protected]
```

Removes the item at the specified line.  Caveat: See efficiency note in find_line(). You must call redraw() to make any changes visible.

**Parameters**

- **in line**  The line number to be removed. (1 based) Must be in range!

**Returns**

Pointer to browser item that was removed (and is no longer valid).

See also

add(), insert(), remove(), swap(int, int), clear()

12.13.3.2  add()

```c
void Fl_Browser::add ( const char * newtext,
                        void * d = 0 )
```

Adds a new line to the end of the browser.  The text string newtext may contain format characters; see format_char() for details.  newtext is copied using the strdup() function, and can be NULL to make a blank line.  The optional void* argument d will be the data() for the new item.

**Parameters**

- **in newtext**  The label text used for the added item
- **in d**  Optional user data() for the item (0 if unspecified)

See also

add(), insert(), remove(), swap(int, int), clear()

12.13.3.3  bottomline()

```c
void Fl_Browser::bottomline ( int line ) [inline]
```

Scrolls the browser so the bottom item in the browser is showing the specified line.

**Parameters**

- **in line**  The line to be displayed at the bottom.

See also

topline(), middleline(), bottomline(), displayed(), lineposition()

12.13.3.4  clear()

```c
void Fl_Browser::clear ( )
```


Removes all the lines in the browser.

See also

add(), insert(), remove(), swap(int,int), clear()

12.13.3.5 column_char() [1/2]

char Fl_Browser::column_char ( ) const [inline]

Gets the current column separator character. The default is 't' (tab).

See also

column_char(), column_widths()

12.13.3.6 column_char() [2/2]

void Fl_Browser::column_char ( char c ) [inline]

Sets the column separator to c. This will only have an effect if you also set column_widths(). The default is 't' (tab).

See also

column_char(), column_widths()

12.13.3.7 column_widths() [1/2]

const int ∗ Fl_Browser::column_widths ( ) const [inline]

Gets the current column width array. This array is zero-terminated and specifies the widths in pixels of each column. The text is split at each column_char() and each part is formatted into its own column. After the last column any remaining text is formatted into the space between the last column and the right edge of the browser, even if the text contains instances of column_char() . The default value is a one-element array of just a zero, which means there are no columns.

Example:

```cpp
Fl_Browser ∗b = new Fl_Browser(...);
static int widths[] = { 50, 50, 50, 70, 70, 40, 70, 70, 70, 50, 0 }; // widths for each column
b->column_widths(widths); // assign array to widget
b->column_char('\t');    // use tab as the column character
b->add("USER	CPU	MEM	VSZ	RSS	TTY
STAT	TIME	COMMAND");
b->add("root
2888	0.0	0.0	1352	0
tty3
SW	Aug15
t0:00@b@f/sbin/mingetty tty3");
b->add("root
13115	0.0	0.0	1352	0
tty2	nAug30
t0:00@b@f/sbin/mingetty tty2");
...
```

See also

column_char(), column_widths()

12.13.3.8 column_widths() [2/2]

void Fl_Browser::column_widths ( const int ∗ arr ) [inline]

Sets the current array to arr. Make sure the last entry is zero.

See also

column_char(), column_widths()
### 12.13.3.9 data() [1/2]

```cpp
void * Fl_Browser::data ( int line ) const
```

Returns the user data() for specified line. Return value can be NULL if line is out of range or no user data() was defined. The parameter line is 1 based (1 will be the first item in the list).
Parameters

| in  | line  | The line number of the item whose data() is returned. (1 based) |

Returns

The user data pointer (can be NULL)

### 12.13.3.10 data() [2/2]

void Fl_Browser::data (  
    int line,  
    void * d  
)

Sets the user data for specified line to d. Does nothing if line is out of range.

Parameters

| in  | line | The line of the item whose data() is to be changed. (1 based) |
| in  | d    | The new data to be assigned to the item. (can be NULL) |

### 12.13.3.11 display()

void Fl_Browser::display (  
    int line,  
    int val = 1  
)

For back compatibility. This calls show(line) if val is true, and hide(line) otherwise. If val is not specified, the default is 1 (makes the line visible).

See also

show(int), hide(int), display(), visible(), make_visible()

### 12.13.3.12 displayed()

int Fl_Browser::displayed (  
    int line ) const  

Returns non-zero if line has been scrolled to a position where it is being displayed. Checks to see if the item’s vertical position is within the top and bottom edges of the display window. This does NOT take into account the hide()/show() status of the widget or item.

Parameters

| in  | line | The line to be checked |

Returns

1 if visible, 0 if not visible.
See also

topl ine(), middleline(), bottomline(), displayed(), lineposition()

12.13.3.13 find_line()

FL_BLINE * Fl_Browser::find_line (int line) const [protected]

Returns the item for specified line.

Note: This call is slow. It’s fine for e.g. responding to user clicks, but slow if called often, such as in a tight sorting loop. Finding an item ‘by line’ involves a linear lookup on the internal linked list. The performance hit can be significant if the browser’s contents is large, and the method is called often (e.g. during a sort). If you’re writing a subclass, use the protected methods item_first(), item_next(), etc. to access the internal linked list more efficiently.

Parameters

| in  | line          | The line number of the item to return. (1 based) |

Return values

<table>
<thead>
<tr>
<th>item</th>
<th>that was found.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>if line is out of range.</td>
</tr>
</tbody>
</table>

See also

item_at(), find_line(), lineno()

12.13.3.14 format_char () [1/2]

c char Fl_Browser::format_char ( ) const [inline]

Gets the current format code prefix character, which by default is ‘@’.

A string of formatting codes at the start of each column are stripped off and used to modify how the rest of the line is printed:

- ’@.’ Print rest of line, don’t look for more ‘@’ signs
- ’@@’ Doubling the format character prints the format character once, followed by the rest of line
- ’@l’ Use a LARGE (24 point) font
- ’@m’ Use a medium large (18 point) font
- ’@s’ Use a small (11 point) font
- ’@b’ Use a bold font (adds FL_BOLD to font)
- ’@i’ Use an italic font (adds FL_ITALIC to font)
- ’@f’ or ’@t’ Use a fixed-pitch font (sets font to FL_COURIER)
- ’@c’ Center the line horizontally
- ’@r’ Right-justify the text
- ’@N’ Use fl_inactive_color() to draw the text
- ’@B0’, ’@B1’, ... ’@B255’ Fill the background with fl_color(n)
- ’@C0’, ’@C1’, ... ’@C255’ Use fl_color(n) to draw the text
- ’@F0’, ’@F1’, ... Use fl_font(n) to draw the text
• '@S1', '@S2', ... Use point size n to draw the text
• '@u' or '@_ Underline the text.
• '@-' draw an engraved line through the middle.

Notice that the '@.' command can be used to reliably terminate the parsing. To print a random string in a random color, use `sprintf("@C%d@.%s", color, string)` and it will work even if the string starts with a digit or has the format character in it.

12.13.3.15 format_char() [2/2]
void Fl_Browser::format_char (char c) [inline]
Sets the current format code prefix character to c.
The default prefix is '@'. Set the prefix to 0 to disable formatting.

See also
format_char() for list of '@' codes

12.13.3.16 full_height()
int Fl_Browser::full_height ( ) const [protected], [virtual]
The height of the entire list of all visible() items in pixels.
This returns the accumulated height of all the items in the browser that are not hidden with hide(), including items scrolled off screen.

Returns
The accumulated size of all the visible items in pixels.

See also
item_height(), item_width(), incr_height(), full_height()
Reimplemented from Fl_Browser_.

12.13.3.17 hide() [1/2]
void Fl_Browser::hide ( ) [inline], [virtual]
Hides the entire Fl_Browser widget – opposite of show().
Reimplemented from Fl_Widget.

12.13.3.18 hide() [2/2]
void Fl_Browser::hide (int line )
Makes line invisible, preventing selection by the user.
The line can still be selected under program control. This changes the full_height() if the state was changed. When a line is made invisible, lines below it are moved up in the display. redraw() is called automatically if a change occurred.

Parameters

| in | line | The line to be hidden. (1 based) |
See also

   show(int), hide(int), display(), visible(), make_visible()

12.13.3.19 icon() [1/2]

Fl_Image * Fl_Browser::icon ( int line ) const

Returns the icon currently defined for line. If no icon is defined, NULL is returned.

Parameters

   in   line  The line whose icon is returned.

Returns

   The icon defined, or NULL if none.

12.13.3.20 icon() [2/2]

void Fl_Browser::icon ( int line, Fl_Image * icon )

Set the image icon for line to the value icon. Caller is responsible for keeping the icon allocated. The line is automatically redrawn.

Parameters

   in   line  The line to be modified. If out of range, nothing is done.
   in   icon  The image icon to be assigned to the line. If NULL, any previous icon is removed.

12.13.3.21 incr_height()

int Fl_Browser::incr_height ( ) const [protected], [virtual]

The default 'average' item height (including inter-item spacing) in pixels. This currently returns textsizer() + 2.

Returns

   The value in pixels.

See also

   item_height(), item_width(), incr_height(), full_height()

Reimplemented from Fl_Browser_.

12.13.3.22 insert() [1/2]

void Fl_Browser::insert ( int line, const char * newtext, void * d = 0 )

Insert a new entry whose label is newtext above given line, optional data d. Text may contain format characters; see format_char() for details. newtext is copied using the strdup() function, and can be NULL to make a blank line. The optional void * argument d will be the data() of the new item.
12.13 Fl_Browser Class Reference

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>Line position for insert. (1 based)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>If line &gt; size(), the entry will be added at the end.</td>
</tr>
<tr>
<td>in</td>
<td>newtext</td>
<td>The label text for the new line.</td>
</tr>
<tr>
<td>in</td>
<td>d</td>
<td>Optional pointer to user data to be associated with the new line.</td>
</tr>
</tbody>
</table>

12.13.3.23 insert() [2/2]

```c
void Fl_Browser::insert (  
    int line,  
    FL_BLINE * item ) [protected]
```

Insert specified item above line.  
If line > size() then the line is added to the end.  
Caveat: See efficiency note in find_line().

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>The new line will be inserted above this line (1 based).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>item</td>
<td>The item to be added.</td>
</tr>
</tbody>
</table>

12.13.3.24 item_at()

```c
void * Fl_Browser::item_at (  
    int line ) const [inline], [protected], [virtual]
```

Return the item at specified line.

Parameters

| in   | line    | The line of the item to return. (1 based) |

Returns

The item, or NULL if line out of range.

See also

- item_at(), find_line(), lineno()

Reimplemented from Fl_Browser_.

12.13.3.25 item_draw()

```c
void Fl_Browser::item_draw (  
    void * item,  
    int X,  
    int Y,  
    int W,  
    int H ) const [protected], [virtual]
```

Draws item at the position specified by X Y W H.  
The W and H values are used for clipping. Should only be called within the context of an FLTK draw().

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item to be drawn</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y,W,H</td>
<td>position and size.</td>
</tr>
</tbody>
</table>
Implements `Fl_Browser_`.

### 12.13.3.26 item_first()

```cpp
void * Fl_Browser::item_first ( ) const [protected], [virtual]
```

Returns the very first item in the list.

**Example of use:**
```cpp
// Walk the browser from beginning to end
for ( void *i=item_first(); i; i=item_next(i) ) {
    printf("item label='\$s'\n", item_text(i));
}
```

**Returns**

The first item, or NULL if list is empty.

**See also**

- `item_first()`, `item_last()`, `item_next()`, `item_prev()`

Implements `Fl_Browser_`.

### 12.13.3.27 item_height()

```cpp
int Fl_Browser::item_height ( void * item ) const [protected], [virtual]
```

Returns height of `item` in pixels.

This takes into account embedded @ codes within the `text()` label.

**Parameters**

| in  | item | The item whose height is returned. |

**Returns**

The height of the item in pixels.

**See also**

- `item_height()`, `item_width()`,
- `incr_height()`, `full_height()`

Implements `Fl_Browser_`.

### 12.13.3.28 item_last()

```cpp
void * Fl_Browser::item_last ( ) const [protected], [virtual]
```

Returns the very last item in the list.

**Example of use:**
```cpp
// Walk the browser in reverse, from end to start
for ( void *i=item_last(); i; i=item_prev(i) ) {
    printf("item label='\$s'\n", item_text(i));
}
```

**Returns**

The last item, or NULL if list is empty.

**See also**

- `item_first()`, `item_last()`, `item_next()`, `item_prev()`

Reimplemented from `Fl_Browser_`. 

---

Generated by Doxygen
12.13.3.29  item_next()

void * Fl_Browser::item_next (  
    void * item ) const [protected], [virtual]

Returns the next item after item.

Parameters

| in  | item | The 'current' item |

Returns

The next item after item, or NULL if there are none after this one.

See also

item_first(), item_last(), item_next(), item_prev()

Implements Fl_Browser_.

12.13.3.30  item_prev()

void * Fl_Browser::item_prev (  
    void * item ) const [protected], [virtual]

Returns the previous item before item.

Parameters

| in  | item | The 'current' item |

Returns

The previous item before item, or NULL if there are none before this one.

See also

item_first(), item_last(), item_next(), item_prev()

Implements Fl_Browser_.

12.13.3.31  item_select()

void Fl_Browser::item_select (  
    void * item,  
    int val ) [protected], [virtual]

Change the selection state of item to the value val.

Parameters

| in  | item | The item to be changed. |
| in  | val  | The new selection state: 1 selects, 0 de-selects. |
See also

    select(), selected(), value(), item_select(), item_selected()

Reimplemented from Fl_Browser_.

12.13.3.32 item_selected()

int Fl_Browser::item_selected (  
    void * item ) const [protected], [virtual]

See if item is selected.

Parameters

| in  | item | The item whose selection state is to be checked. |

Returns

    1 if selected, 0 if not.

See also

    select(), selected(), value(), item_select(), item_selected()

Reimplemented from Fl_Browser_.

12.13.3.33 item_swap()

void Fl_Browser::item_swap (  
    void * a,  
    void * b ) [inline], [protected], [virtual]

Swap the items a and b.
You must call redraw() to make any changes visible.

Parameters

| in  | a,b | the items to be swapped. |

See also

    swap(int,int), item_swap()

Reimplemented from Fl_Browser_.

12.13.3.34 item_text()

const char * Fl_Browser::item_text (  
    void * item ) const [protected], [virtual]

Returns the label text for item.

Parameters

| in  | item | The item whose label text is returned. |

Returns

    The item's text string. (Can be NULL)

Reimplemented from Fl_Browser_.

---

Generated by Doxygen
12.13.3.35 item_width()

```c
int Fl_Browser::item_width (  
    void * item ) const [protected], [virtual]
```

Returns width of `item` in pixels. This takes into account embedded `@` codes within the `text()` label.

**Parameters**

| in | item | The item whose width is returned. |

**Returns**

The width of the item in pixels.

**See also**

`item_height()`, `item_width()`, `incr_height()`, `full_height()`

Implements `Fl_Browser_`.

12.13.3.36 lineno()

```c
int Fl_Browser::lineno (  
    void * item ) const [protected]
```

Returns line number corresponding to `item`, or zero if not found. Caveat: See efficiency note in `find_line()`.

**Parameters**

| in | item | The item to be found |

**Returns**

The line number of the item, or 0 if not found.

**See also**

`item_at()`, `find_line()`, `lineno()`

12.13.3.37 lineposition()

```c
void Fl_Browser::lineposition (  
    int line,  
    Fl_Line_Position pos)
```

Updates the browser so that `line` is shown at position `pos`.

**Parameters**

| in | line | line number. (1 based) |
| in | pos  | position. |

**See also**

`topline()`, `middleline()`, `bottomline()`
12.13.3.38  load()

int Fl_Browser::load ( const char * filename )

Clears the browser and reads the file, adding each line from the file to the browser.
If the filename is NULL or a zero-length string then this just clears the browser. This returns zero if there was any
error in opening or reading the file, in which case errno is set to the system error. The data() of each line is set to
NULL.

Parameters

- **filename** The filename to load

Returns

1 if OK, 0 on error (errno has reason)

See also

add()

12.13.3.39  make_visible()

void Fl_Browser::make_visible ( int line ) [inline]

Make the item at the specified line visible().
Functionally similar to show(int line). If line is out of range, redisplay top or bottom of list as appropriate.

Parameters

- **line** The line to be made visible.

See also

show(int), hide(int), display(), visible(), make_visible()

12.13.3.40  middleline()

void Fl_Browser::middleline ( int line ) [inline]

Scrolls the browser so the middle item in the browser is showing the specified line.

Parameters

- **line** The line to be displayed in the middle.

See also

topline(), middleline(), bottomline(), displayed(), lineposition()

12.13.3.41  move()

void Fl_Browser::move ( int to, int from )

Line from is removed and reinserted at to.
Note: to is calculated after line from gets removed.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>to</th>
<th>Destination line number (calculated after line from is removed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>from</td>
<td>Line number of item to be moved</td>
</tr>
</tbody>
</table>

### 12.13.3.42 remove()

```cpp
void Fl_Browser::remove (int line)
```

Remove entry for given line number, making the browser one line shorter. You must call `redraw()` to make any changes visible.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>Line to be removed. (1 based)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>If line is out of range, no action is taken.</td>
</tr>
</tbody>
</table>

See also

- `add()`, `insert()`, `remove()`, `swap(int,int)`, `clear()`

### 12.13.3.43 remove_icon()

```cpp
void Fl_Browser::remove_icon (int line)
```

Removes the icon for line.

It's ok to remove an icon if none has been defined.

**Parameters**

| in  | line     | The line whose icon is to be removed.                           |

### 12.13.3.44 select()

```cpp
int Fl_Browser::select (int line, int val = 1)
```

Sets the selection state of the item at line to the value val.

If val is not specified, the default is 1 (selects the item).

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>The line number of the item to be changed. (1 based)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>val</td>
<td>The new selection state (1=select, 0=de-select).</td>
</tr>
</tbody>
</table>

**Returns**

1 if the state changed, 0 if not.

See also

- `select()`, `selected()`, `value()`, `item_select()`, `item_selected()`
12.13.3.45 **selected()**

```cpp
int Fl_Browser::selected ( int line ) const
```

Returns 1 if specified `line` is selected, 0 if not.

**Parameters**

- `line` (1 based)

**Returns**

1 if item selected, 0 if not.

**See also**

- select(), selected(), value(), item_select(), item_selected()

12.13.3.46 **show() [1/2]**

```cpp
void Fl_Browser::show ( ) [inline], [virtual]
```

Shows the entire Fl_Browser widget – opposite of hide(). Reimplemented from Fl_Widget.

12.13.3.47 **show() [2/2]**

```cpp
void Fl_Browser::show ( int line )
```

Makes `line` visible, and available for selection by user. Opposite of hide(int). This changes the full_height() if the state was changed. redraw() is called automatically if a change occurred.

**Parameters**

- `line` (1 based)

**See also**

- show(int), hide(int), display(), visible(), make_visible()

12.13.3.48 **size()**

```cpp
int Fl_Browser::size ( ) const [inline]
```

Returns how many lines are in the browser. The last line number is equal to this. Returns 0 if browser is empty.

12.13.3.49 **swap() [1/2]**

```cpp
void Fl_Browser::swap ( FL_BLINE ∗ a, FL_BLINE ∗ b ) [protected]
```

Swap the two items `a` and `b`. Uses swapping() to ensure list updates correctly.

**Parameters**

- `a`, `b`
See also

\texttt{swap(int,int), item\_swap()}

\section*{12.13.3.50 \texttt{swap()} [2/2]}

\begin{verbatim}
void Fl_Browser::swap (  
    int a,  
    int b )
\end{verbatim}

Swaps two browser lines \texttt{a} and \texttt{b}. You must call \texttt{redraw()} to make any changes visible.

Parameters

\begin{itemize}
  \item \texttt{in \ a,b} \hspace{1em} The two lines to be swapped. (both 1 based)
\end{itemize}

See also \texttt{swap(int,int), item\_swap()}

\section*{12.13.3.51 \texttt{text()} [1/2]}

\begin{verbatim}
const char * Fl_Browser::text (  
    int line ) const
\end{verbatim}

Returns the label text for the specified \texttt{line}. Return value can be NULL if \texttt{line} is out of range or unset. The parameter \texttt{line} is 1 based.

Parameters

\begin{itemize}
  \item \texttt{in \ line} \hspace{1em} The line number of the item whose text is returned. (1 based)
\end{itemize}

Returns

The text string (can be NULL)

\section*{12.13.3.52 \texttt{text()} [2/2]}

\begin{verbatim}
void Fl_Browser::text (  
    int line,  
    const char * newtext )
\end{verbatim}

Sets the text for the specified \texttt{line} to \texttt{newtext}. Text may contain format characters; see \texttt{format\_char()} for details. \texttt{newtext} is copied using the \texttt{strdup()} function, and can be NULL to make a blank line. Does nothing if \texttt{line} is out of range.

Parameters

\begin{itemize}
  \item \texttt{in \ line} \hspace{1em} The line of the item whose text will be changed. (1 based)
  \item \texttt{in \ newtext} \hspace{1em} The new string to be assigned to the item.
\end{itemize}

\section*{12.13.3.53 \texttt{textsize()}}

\begin{verbatim}
void Fl_Browser::textsize (  
    Fl_Fontsize newSize )
\end{verbatim}

Sets the default text size (in pixels) for the lines in the browser to \texttt{newSize}.
This method recalculates all item heights and caches the total height internally for optimization of later item changes. This can be slow if there are many items in the browser. It returns immediately (w/o recalculation) if newSize equals the current textsize(). You may need to call redraw() to see the effect and to have the scrollbar positions recalculated. You should set the text size before populating the browser with items unless you really need to change the size later.

12.13.3.54 topline() [1/2]
int Fl_Browser::topline ( ) const
Returns the line that is currently visible at the top of the browser. If there is no vertical scrollbar then this will always return 1.

Returns
The lineno() of the top() of the browser.

12.13.3.55 topline() [2/2]
void Fl_Browser::topline ( int line ) [inline]
Scrolls the browser so the top item in the browser is showing the specified line.

Parameters

<table>
<thead>
<tr>
<th>In</th>
<th>line</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The line to be displayed at the top.</td>
</tr>
</tbody>
</table>

See also
toline(), middleline(), bottomline(), displayed(), lineposition()

12.13.3.56 value() [1/2]
int Fl_Browser::value ( ) const
Returns the line number of the currently selected line, or 0 if none selected.

Returns
The line number of current selection, or 0 if none selected.

See also
select(), selected(), value(), item_select(), item_selected()

12.13.3.57 value() [2/2]
void Fl_Browser::value ( int line ) [inline]
Sets the browser’s value(), which selects the specified line. This is the same as calling select(line).

See also
select(), selected(), value(), item_select(), item_selected()

12.13.3.58 visible()
int Fl_Browser::visible ( int line ) const
Returns non-zero if the specified line is visible, 0 if hidden. Use show(int), hide(int), or make_visible(int) to change an item’s visible state.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>line</td>
<td>The line in the browser to be tested. (1 based)</td>
</tr>
</tbody>
</table>

See also

- `show(int)`, `hide(int)`, `display()`, `visible()`, `make_visible()`

The documentation for this class was generated from the following files:

- `Fl_Browser.H`
- `Fl_Browser.cxx`
- `Fl_Browser_load.cxx`

### 12.14 Fl_Browser Class Reference

This is the base class for browsers.

```cpp
#include <Fl_Browser_.H>
```

Inheritance diagram for Fl_Browser_:

```
Fl_Widget
    └── Fl_Group
        └── Fl_Browser
            └── Fl_Browser
                ├── Fl_Browser
                │    └── Fl_Browser
                ├── Fl_Check_Browser
                └──...
```

#### Public Types

- `enum { HORIZONTAL = 1, VERTICAL = 2, BOTH = 3, ALWAYS_ON = 4, HORIZONTAL_ALWAYS = 5, VERTICAL_ALWAYS = 6, BOTH_ALWAYS = 7 }`

  Values for `has_scrollbar()`.

#### Public Member Functions

- `int deselect (int docallbacks=0)`
  
  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.

- `void display (void *item)`
  
  Displays the item, scrolling the list as necessary.

- `int handle (int event) FL_OVERRIDE`
  
  Handles the event within the normal widget bounding box.

- `uchar has_scrollbar () const`
  
  Returns the current scrollbar mode, see `Fl_Browser_::has_scrollbar(uchar)`

- `void has_scrollbar (uchar mode)`
  
  Sets whether the widget should have scrollbars or not (default `Fl_Browser_::BOTH`).

- `int hposition () const`
  
  Gets the horizontal scroll position of the list as a pixel position `pos`.

- `void hposition (int)`

Generated by Doxygen
Sets the horizontal scroll position of the list to pixel position \( \text{pos} \).

- **int linespacing () const**
  
  Return the height of additional spacing between browser lines.

- **void linespacing (int pixels)**
  
  Add some space between browser lines.

- **int position () const**

- **void position (int pos)**

- **void position (int \( x \), int \( y \))**

- **void resize (int \( X \), int \( Y \), int \( W \), int \( H \)) FL_OVERRIDE**
  
  Repositions and/or resizes the browser.

- **void scrollbar_left ()**
  
  Moves the vertical scrollbar to the lefthand side of the list.

- **void scrollbar_right ()**
  
  Moves the vertical scrollbar to the righthand side of the list.

- **int scrollbar_size () const**

- **void scrollbar_size (int new\( \text{Size} \))**
  
  Sets the pixel size of the scrollbars' troughs to new\( \text{Size} \), in pixels.

- **int scrollbar_width () const**
  
  Returns the global value Fl::scrollbar_size().

- **void scrollbar_width (int width)**
  
  Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

- **int select (void *\( \text{item} \), int val=1, int docallbacks=0)**
  
  Sets the selection state of \( \text{item} \) to \( \text{val} \), and returns 1 if the state changed or 0 if it did not.

- **int select_only (void *\( \text{item} \), int docallbacks=0)**
  
  Selects \( \text{item} \) and returns 1 if the state changed or 0 if it did not.

- **void sort (int flags=0)**
  
  Sort the items in the browser based on \( \text{flags} \).

- **Fl_Color textcolor () const**
  
  Gets the default text color for the lines in the browser.

- **void textcolor (Fl_Color \( \text{col} \))**
  
  Sets the default text color for the lines in the browser to \( \text{col} \).

- **Fl_Font textfont () const**
  
  Gets the default text font for the lines in the browser.

- **void textfont (Fl_Font \( \text{font} \))**
  
  Sets the default text font for the lines in the browser to \( \text{font} \).

- **Fl_Fontsize textsize () const**
  
  Gets the default text size (in pixels) for the lines in the browser.

- **void textsize (Fl_Fontsize new\( \text{Size} \))**
  
  Sets the default text size (in pixels) for the lines in the browser to \( \text{size} \).

- **int vposition () const**
  
  Gets the vertical scroll position (in pixels) for the lines in the browser.

- **void vposition (int \( \text{pos} \))**
  
  Sets the vertical scroll position of the list as a pixel position \( \text{pos} \).

- **void vposition (int \( \text{pos} \))**
  
  Sets the vertical scroll position of the list to pixel position \( \text{pos} \).
Public Member Functions inherited from Fl_Group

- Fl_Widget * _ddfdesign_kludge ()
  This is for forms compatibility only.
- void add (Fl_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.
- void add (Fl_Widget *)
  See void Fl_Group::add(Fl_Widget &w)
- void add_resizable (Fl_Widget &o)
  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const
  Returns a pointer to the array of children.
- Fl_Group const * as_group () const FL_OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()
  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const
  Returns array()[n].
- int children () const
  Returns how many child widgets the group has.
- void clear ()
  Deletes all child widgets from memory recursively.
- unsigned int clip_children ()
  Returns the current clipping mode.
- void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.
- void end ()
  Exactly the same as current(this->parent()).
- int find (const Fl_Widget &o) const
  See int Fl_Group::find(const Fl_Widget *w) const.
- int find (const Fl_Widget *) const
  Searches the child array for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *)=0
  Creates a new Fl_Group widget using the given position, size, and label string.
- void focus (Fl_Widget *)
- void forms_end ()
  This is for forms compatibility only.
- int handle (int) FL_OVERRIDE
  Handles the specified event.
- void init_sizes ()
  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &, Fl_Widget *)
  This does insert(w, find(before)).
- void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *)
Removes the widget from the group.

• void remove (int index)
  Removes the widget at index from the group but does not delete it.

• Fl_Widget ∗ resizable () const
  Returns the group’s resizable widget.

• void resizable (Fl_Widget &o)
  Sets the group’s resizable widget.

• void resizable (Fl_Widget ∗o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

• void resize (int, int, int, int) FL_OVERRID
  Resizes the Fl_Group widget and all of its children.

• virtual ~Fl_Group ()
  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  Activates the widget.

• unsigned int active () const
  Returns whether the widget is active.

• int active_r () const
  Returns whether the widget and all of its parents are active.

• Fl_Align align () const
  Gets the label alignment.

• void align (Fl_Align alignment)
  Sets the label alignment.

• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.

• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const ∗ as_gl_window () const
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window const ∗ as_window () const
• void bind_deimage (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the inactive state.

• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

• void bind_image (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the active state.

• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

• Fl_Boxtype box () const
  Gets the box type of the widget.

• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.

• Fl_Callback_p callback () const
Gets the current callback function for the widget.

- void callback (Fl_Callback ∗cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback ∗cb, void ∗p)
  Sets the current callback function and data for the widget.

- void callback (Fl_Callback0 ∗cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback1 ∗cb, long p=0)
  Sets the current callback function for the widget.

- unsigned int changed () const
  Checks if the widget value changed since the last callback.

- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  Clears or sets the damage flags.

- void clear_output ()
  Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.

- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

- Fl_Color color () const
  Gets the background color of the widget.

- void color (Fl_Color bg)
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

- Fl_Color color2 () const
  For back compatibility only.

- void color2 (unsigned a)
  For back compatibility only.

- int contains (const Fl_Widget ∗w) const
  Checks if w is a child of this widget.

- void copy_label (const char ∗new_label)
  Sets the current label.

- void copy_tooltip (const char ∗text)
  Sets the current tooltip text.

- uchar damage () const
  Returns non-zero if draw() needs to be called.

- void damage (uchar c)
  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- int damage_resize (int, int, int, int)
  Internal use only.

- void deactivate ()
  Deactivates the widget.
• **FL_Image * deimage ()**
  
  *Gets the image that is used as part of the widget label when in the inactive state.*

• **const FL_Image * deimage () const**
  
  *Gets the image that is used as part of the widget label when in the inactive state.*

• **void deimage (FL_Image &img)**
  
  *Sets the image to use as part of the widget label when in the inactive state.*

• **void deimage (FL_Image *img)**
  
  *Sets the image to use as part of the widget label when in the inactive state.*

• **int deimage_bound () const**
  
  *Returns whether the inactive image is managed by the widget.*

• **void do_callback (FI_Callback_Reason reason=FL_REASON_UNKNOWN)**
  
  *Calls the widget callback function with default arguments.*

• **void do_callback (FI_Widget *widget, long arg, FI_Callback_Reason reason=FL_REASON_UNKNOWN)**
  
  *Calls the widget callback function with arbitrary arguments.*

• **void do_callback (FI_Widget *widget, void *arg=0, FI_Callback_Reason reason=FL_REASON_UNKNOWN)**
  
  *Calls the widget callback function with arbitrary arguments.*

• **void draw_label (int, int, int, int, FI_Align) const**
  
  *Draws the label in an arbitrary bounding box with an arbitrary alignment.*

• **int h () const**
  
  *Gets the widget height.*

• **virtual void hide ()**
  
  *Makes a widget invisible.*

• **FL_Image * image ()**
  
  *Gets the image that is used as part of the widget label when in the active state.*

• **const FL_Image * image () const**
  
  *Gets the image that is used as part of the widget label when in the active state.*

• **void image (FL_Image &img)**
  
  *Sets the image to use as part of the widget label when in the active state.*

• **void image (FL_Image *img)**
  
  *Sets the image to use as part of the widget label when in the active state.*

• **int image_bound () const**
  
  *Returns whether the image is managed by the widget.*

• **int inside (const FL_Widget *wgt) const**
  
  *Checks if this widget is a child of wgt.*

• **int is_label_copied () const**
  
  *Returns whether the current label was assigned with copy_label().*

• **const char * label () const**
  
  *Gets the current label text.*

• **void label (const char *text)**
  
  *Sets the current label pointer.*

• **void label (FI_Labeltype a, const char *b)**
  
  *Shortcut to set the label text and type in one call.*

• **FL_Color labelcolor () const**
  
  *Gets the label color.*

• **void labelcolor (FL_Color c)**
  
  *Sets the label color.*

• **FL_Font labelfont () const**
  
  *Gets the font to use.*

• **void labelfont (FL_Font f)**
  
  *Sets the font to use.*

• **FI_Fontsize labelsize () const**
Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '§' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '§' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.

• const char * tooltip () const
  Gets the current tooltip text.

• void tooltip (const char *text)
  Sets the current tooltip text.

• Fl_Widget * top_window () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const
  Gets the widget type.

• void type (uchar t)
  Sets the widget type.

• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void * user_data () const
  Gets the user data for this widget.

• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

• void user_data (void *v)
  Sets the user data for this widget.

• unsigned int visible () const
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

• void visible_focus (int v)
  Modifies keyboard focus navigation.

• int visible_r () const
  Returns whether a widget and all its parents are visible.

• int w () const
  Gets the widget width.

• Fl_When when () const
  Returns the conditions under which the callback is called.

• void when (uchar i)
  Sets the flags used to decide when a callback is called.

• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  Gets the widget position in its window.

• int y () const
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  Destroys the widget.

Public Attributes

• Fl_Scrollbar hscrollbar
  Horizontal scrollbar.

• Fl_Scrollbar scrollbar
  Vertical scrollbar.
Protected Member Functions

- void **bbox** (int &X, int &Y, int &W, int &H) const
  
  Returns the bounding box for the interior of the list's display window, inside the scrollbars.

- void **deleting** (void *item)
  
  This method should be used when item is being deleted from the list.

- int **displayed** (void *item) const
  
  Returns non-zero if item has been scrolled to a position where it is being displayed.

- void **draw** () FL_OVERRIDE
  
  Draws the list within the normal widget bounding box.

- void *find_item (int ypos)
  
  This method returns the item under mouse y position ypos.

- **Fl_Browser_** (int X, int Y, int W, int H, const char *L=0)
  
  The constructor makes an empty browser.

- virtual int **full_height** () const
  
  This method may be provided by the subclass to indicate the full height of the item list, in pixels.

- virtual int **full_width** () const
  
  This method may be provided by the subclass to indicate the full width of the item list, in pixels.

- virtual int **incr_height** () const
  
  This method may be provided to return the average height of all items to be used for scrolling.

- void **inserting** (void *a, void *b)
  
  This method should be used when an item is in the process of being inserted into the list.

- virtual void *item_at (int index) const
  
  This method must be provided by the subclass to return the item for the specified index.

- virtual void **item_draw** (void *item, int X, int Y, int W, int H) const =0
  
  This method must be provided by the subclass to draw the item in the area indicated by X, Y, W, H.

- virtual void *item_first () const =0
  
  This method must be provided by the subclass to return the first item in the list.

- virtual int **item_height** (void *item) const =0
  
  This method must be provided by the subclass to return the height of item in pixels.

- virtual void *item_last () const
  
  This method must be provided by the subclass to return the last item in the list.

- virtual void *item_next (void *item) const =0
  
  This method must be provided by the subclass to return the item in the list after item.

- virtual void *item_prev (void *item) const =0
  
  This method must be provided by the subclass to return the item in the list before item.

- virtual int **item_quick_height** (void *item) const
  
  This method may be provided by the subclass to return the height of the item, in pixels.

- virtual void **item_select** (void *item, int val=1)
  
  This method must be implemented by the subclass if it supports multiple selections; sets the selection state to val for the item.

- virtual int **item_selected** (void *item) const
  
  This method must be implemented by the subclass if it supports multiple selections; returns the selection state for item.

- virtual void **item_swap** (void *a, void *b)
  
  This optional method should be provided by the subclass to efficiently swap browser items a and b, such as for sorting.

- virtual const char * **item_text** (void *item) const
  
  This optional method returns a string (label) that may be used for sorting.

- virtual int **item_width** (void *item) const =0
  
  This method must be provided by the subclass to return the width of the item in pixels.
This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.

- **void new_list ()**
  This method should be called when the list data is completely replaced or cleared.

- **void redraw_line (void *item)**
  This method should be called when the contents of item has changed, but not its height.

- **void redraw_lines ()**
  This method will cause the entire list to be redrawn.

- **void replacing (void *a, void *b)**
  This method should be used when item a is being replaced by item b.

- **void *selection () const**
  Returns the item currently selected, or NULL if there is no selection.

- **void swapping (void *a, void *b)**
  This method should be used when two items a and b are being swapped.

- **void *top () const**
  Returns the item that appears at the top of the list.

Protected Member Functions inherited from Fl_Group

- **Fl_Rect *bounds ()**
  Returns the internal array of widget sizes and positions.

- **void draw () FL_OVERRIDE**
  Draws the widget.

- **void draw_child (Fl_Widget &widget) const**
  Forces a child to redraw.

- **void draw_children ()**
  Draws all children of the group.

- **void draw_outside_label (const Fl_Widget &widget) const**
  Parents normally call this to draw outside labels of child widgets.

- **virtual int on_insert (Fl_Widget *, int)**
  Allow derived groups to act when a widget is added as a child.

- **virtual int on_move (int, int)**
  Allow derived groups to act when a widget is moved within the group.

- **virtual void on_remove (int)**
  Allow derived groups to act when a child widget is removed from the group.

- **int *sizes ()**
  Returns the internal array of widget sizes and positions.

- **void update_child (Fl_Widget &widget) const**
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- **void clear_flag (unsigned int c)**
  Clears a flag in the flags mask.

- **void draw_backdrop () const**
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- **void draw_box () const**
  Draws the widget box according its box style.

- **void draw_box (Fl_Boxtype t, Fl_Color c) const**
  Draws a box of type t, of color c at the widget's position and size.

- **void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const**
  Draws a box of type t, of color c at the position X,Y and size W,H.

- **void draw_focus () const**

Generated by Doxygen
**Draws a focus rectangle around the widget.**

- `void draw_focus(Fl_Widget t, int X, int Y, int W, int H) const`
  
  Draws a focus rectangle around the widget.

- `void draw_focus(Fl_Widget t, int x, int y, int w, int h, Fl_Color bg) const`
  
  Draws a focus box for the widget at the given position and size.

- `void draw_label() const`
  
  Draws the widget's label at the defined label position.

- `void draw_label(int, int, int, int) const`
  
  Draws the label in an arbitrary bounding box.

- `Fl_Widget(int x, int y, int w, int h, const char *label=0L)`
  
  Creates a widget at the given position and size.

- `unsigned int flags() const`
  
  Gets the widget flags mask.

- `void h(int v)`
  
  Internal use only.

- `void set_flag(unsigned int c)`
  
  Sets a flag in the flags mask.

- `void w(int v)`
  
  Internal use only.

- `void x(int v)`
  
  Internal use only.

- `void y(int v)`
  
  Internal use only.

### Additional Inherited Members

**Static Public Member Functions inherited from Fl_Group**

- `static Fl_Group * current()`
  
  Returns the currently active group.

- `static void current(Fl_Group *g)`
  
  Sets the current group.

**Static Public Member Functions inherited from Fl_Widget**

- `static void default_callback(Fl_Widget *widget, void *data)`
  
  The default callback for all widgets that don't set a callback.

- `static unsigned int label_shortcut(const char *t)`
  
  Returns the Unicode value of the '&x' shortcut in a given text.

- `static int test_shortcut(const char *t, const bool require_alt=false)`
  
  Returns true if the given text t contains the entered '&x' shortcut.

**Protected Types inherited from Fl_Widget**

- `enum {
  INACTIVE = 1<10, INVISIBLE = 1<11, OUTPUT = 1<12, NOBORDER = 1<13,
  FORCE_POSITION = 1<14, NON_MODAL = 1<15, SHORTCUT_LABEL = 1<16, CHANGED = 1<17,
  OVERRIDE = 1<18, VISIBLE_FOCUS = 1<19, COPIED_LABEL = 1<20, CLIP_CHILDREN = 1<21,
  MENU_WINDOW = 1<22,_TOOLTIP_WINDOW = 1<23, MODAL = 1<24, NO_OVERLAY = 1<25,
  GROUP_RELATIVE = 1<26, COPIED_TOOLTIP = 1<27, FULLSCREEN = 1<28, MAC_USE_ACCENTS_MENU = 1<29,
} Generated by Doxygen
NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
AUTO_DELETE_USER_DATA = 1<<23 ,
MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 \}

flags possible values enumeration.

12.14.1 Detailed Description

This is the base class for browsers. To be useful it must be subclassed and several virtual functions defined. The Forms-compatible browser and the file chooser's browser are subclassed off of this. This has been designed so that the subclass has complete control over the storage of the data, although because next() and prev() functions are used to index, it works best as a linked list or as a large block of characters in which the line breaks must be searched for. A great deal of work has been done so that the "height" of a data object does not need to be determined until it is drawn. This is useful if actually figuring out the size of an object requires accessing image data or doing stat() on a file or doing some other slow operation. Callbacks are called when the value changes with FL_REASON_CHANGED. If FL_WHEN_RELEASE is set, callbacks are called when the mouse button is released with FL_REASON_CHANGED or FL_REASON_RESELECTED if the selection did not change. If FL_WHEN_ENTER_KEY is set, callbacks are also called when key presses or double clicks change the selection.

Keyboard navigation of browser items

The keyboard navigation of browser items is only possible if visible_focus() is enabled. If disabled, the widget rejects keyboard focus; Tab and Shift-Tab focus navigation will skip the widget. In 'Select' and 'Normal' mode, the widget rejects keyboard focus; no navigation keys are supported (other than scrollbar positioning). In 'Hold' mode, the widget accepts keyboard focus, and Up/Down arrow keys can navigate the selected item. In 'Multi' mode, the widget accepts keyboard focus, and Up/Down arrow keys navigate the focus box; Space toggles the current item's selection, Enter selects only the current item (deselects all others). If Shift (or Ctrl) is combined with Up/Down arrow keys, the current item's selection state is extended to the next item. In this way one can extend a selection or de-selection.

12.14.2 Member Enumeration Documentation

12.14.2.1 anonymous enum

Anonymous enum bit flags for has_scrollbar().

Anonymous enum

Values for has_scrollbar().

Anonymous enum bit flags for has_scrollbar().

- bit 0: horizontal
- bit 1: vertical
- bit 2: 'always' (to be combined with bits 0 and 1)
- bit 3-31: reserved for future use

Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HORIZONTAL</td>
<td>Only show horizontal scrollbar.</td>
</tr>
<tr>
<td>VERTICAL</td>
<td>Only show vertical scrollbar.</td>
</tr>
<tr>
<td>BOTH</td>
<td>Show both scrollbars. (default)</td>
</tr>
<tr>
<td>ALWAYS_ON</td>
<td>Specified scrollbar(s) should 'always' be shown (to be used with HORIZONTAL/VERTICAL)</td>
</tr>
<tr>
<td>HORIZONTAL_ALWAYS</td>
<td>Horizontal scrollbar always on.</td>
</tr>
<tr>
<td>VERTICAL_ALWAYS</td>
<td>Vertical scrollbar always on.</td>
</tr>
<tr>
<td>BOTH_ALWAYS</td>
<td>Both scrollbars always on.</td>
</tr>
</tbody>
</table>
### Constructor & Destructor Documentation

#### Fl_Browser_()

```cpp
Fl_Browser_::Fl_Browser_ ( 
    int X, 
    int Y, 
    int W, 
    int H, 
    const char ∗L = 0 ) [protected]
```

The constructor makes an empty browser.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>X,Y,W,H</code></td>
<td>position and size.</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>L</code></td>
<td>The label string, may be NULL.</td>
</tr>
</tbody>
</table>

### Member Function Documentation

#### bbox()

```cpp
void Fl_Browser_::bbox ( 
    int ∗X, 
    int ∗Y, 
    int ∗W, 
    int ∗H ) const [protected]
```

Returns the bounding box for the interior of the list's display window, inside the scrollbars.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>X,Y,W,H</code></td>
<td>The returned bounding box. (The original contents of these parameters are overwritten)</td>
</tr>
</tbody>
</table>

#### deleting()

```cpp
void Fl_Browser_::deleting ( 
    void ∗item ) [protected]
```

This method should be used when `item` is being deleted from the list.
It allows the `Fl_Browser_` to discard any cached data it has on the item. This method does not actually delete the item, but handles the follow up bookkeeping after the item has just been deleted.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>item</code></td>
<td>The item being deleted.</td>
</tr>
</tbody>
</table>

#### deselect()

```cpp
int Fl_Browser_::deselect ( 
    int docallbacks = 0 )
```

Deselects all items in the list and returns 1 if the state changed or 0 if it did not.
If the optional `docallbacks` parameter is non-zero, deselect tries to call the callback function for the widget.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>docallbacks</code></td>
<td>If non-zero, invokes widget callback if item changed. If 0, doesn't do callback (default).</td>
</tr>
</tbody>
</table>
12.14.4.4  display()

```cpp
void Fl_Browser_::display ( void * item )
```

Displays the item, scrolling the list as necessary.

**Parameters**

- **item**: The item to be displayed.

**See also**

- display(), displayed()

12.14.4.5  displayed()

```cpp
int Fl_Browser_::displayed ( void * item ) const [protected]
```

Returns non-zero if item has been scrolled to a position where it is being displayed. Checks to see if the item's vertical position is within the top and bottom edges of the display window. This does NOT take into account the hide()/show() status of the widget or item.

**Parameters**

- **item**: The item to check

**Returns**

1 if visible, 0 if not visible.

**See also**

- display(), displayed()

12.14.4.6  draw()

```cpp
void Fl_Browser_::draw ( void ) [protected], [virtual]
```

Draws the list within the normal widget bounding box. Implements Fl_Widget.

12.14.4.7  find_item()

```cpp
void * Fl_Browser_::find_item ( int ypos ) [protected]
```

This method returns the item under mouse y position ypos. NULL is returned if no item is displayed at that position.

**Parameters**

- **ypos**: The y position (eg. Fl::event_y()) to find an item under.

**Returns**

- The item, or NULL if not found
12.14.4.8 full_height()

```cpp
int Fl_Browser_::full_height ( ) const [protected], [virtual]
```

This method may be provided by the subclass to indicate the full height of the item list, in pixels. The default implementation computes the full height from the item heights. Includes the items that are scrolled off screen.

Returns

The height of the entire list, in pixels.

Reimplemented in `Fl_Browser`.

12.14.4.9 full_width()

```cpp
int Fl_Browser_::full_width ( ) const [protected], [virtual]
```

This method may be provided by the subclass to indicate the full width of the item list, in pixels. The default implementation computes the full width from the item widths.

Returns

The maximum width of all the items, in pixels.

12.14.4.10 handle()

```cpp
int Fl_Browser_::handle ( int event ) [virtual]
```

Handles the `event` within the normal widget bounding box.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>event</th>
<th>The event to process.</th>
</tr>
</thead>
</table>

Returns

1 if `event` was processed, 0 if not.

Reimplemented from `Fl_Widget`.
Reimplemented in `Fl_Check_Browser`.

12.14.4.11 has_scrollbar()

```cpp
void Fl_Browser_::has_scrollbar ( uchar mode ) [inline]
```

Sets whether the widget should have scrollbars or not (default `Fl_Browser_::BOTH`). By default you can scroll in both directions, and the scrollbars disappear if the data will fit in the widget. `has_scrollbar()` changes this based on the value of `mode`:

- 0 - No scrollbars.
- `Fl_Browser_::HORIZONTAL` - Only a horizontal scrollbar.
- `Fl_Browser_::VERTICAL` - Only a vertical scrollbar.
- `Fl_Browser_::BOTH` - The default is both scrollbars.
- `Fl_Browser_::HORIZONTAL_ALWAYS` - Horizontal scrollbar always on, vertical always off.
- `Fl_Browser_::VERTICAL_ALWAYS` - Vertical scrollbar always on, horizontal always off.
- `Fl_Browser_::BOTH_ALWAYS` - Both always on.
12.14.4.12 `hposition()` [1/2]

```c
int Fl_Browser_::hposition () const [inline]
```

Gets the horizontal scroll position of the list as a pixel position `pos`. The position returned is how many pixels of the list are scrolled off the left edge of the screen. Example: A position of '18' indicates the left 18 pixels of the list are scrolled off the left edge of the screen.

See also

```c
position(), hposition()
```

12.14.4.13 `hposition()` [2/2]

```c
void Fl_Browser_::hposition ( int pos )
```

Sets the horizontal scroll position of the list to pixel position `pos`. The position is how many pixels of the list are scrolled off the left edge of the screen. Example: A position of '18' scrolls the left 18 pixels of the list off the left edge of the screen.

Parameters

```c
pos
```

The horizontal position (in pixels) to scroll the browser to.

See also

```c
vposition(), hposition()
```

12.14.4.14 `incr_height()`

```c
int Fl_Browser_::incr_height ( ) const [protected], [virtual]
```

This method may be provided to return the average height of all items to be used for scrolling. The default implementation uses the height of the first item.

Returns

The average height of items, in pixels.

Reimplemented in `Fl_Browser`.

12.14.4.15 `inserting()`

```c
void Fl_Browser_::inserting ( void * a, void * b ) [protected]
```

This method should be used when an item is in the process of being inserted into the list. It allows the `Fl_Browser` to update its cache data as needed, scheduling a redraw for the affected lines. This method does not actually insert items, but handles the follow up bookkeeping after items have been inserted.

Parameters

```c
a
b
```

The starting item position

The new item being inserted

12.14.4.16 `item_at()`

```c
virtual void * Fl_Browser_::item_at ( int index ) const [inline], [protected], [virtual]
```

This method must be provided by the subclass to return the item for the specified `index`.
Parameters

- **in index** The index of the item to be returned.

Returns

- The item at the specified index.

Reimplemented in Fl_Check_Browser, and Fl_Browser.

### 12.14.4.17 item_draw()

```cpp
virtual void Fl_Browser_::item_draw ( 
    void * item,
    int X,
    int Y,
    int W,
    int H ) const [protected], [pure virtual]
```

This method must be provided by the subclass to draw the item in the area indicated by X, Y, W, H.
Implemented in Fl_Check_Browser, and Fl_Browser.

### 12.14.4.18 item_first()

```cpp
virtual void * Fl_Browser_::item_first ( ) const [protected], [pure virtual]
```

This method must be provided by the subclass to return the first item in the list.

See also:
- item_first(), item_next(), item_last(), item_prev()  

Implemented in Fl_Browser, and Fl_Check_Browser.

### 12.14.4.19 item_height()

```cpp
virtual int Fl_Browser_::item_height ( 
    void * item ) const [protected], [pure virtual]
```

This method must be provided by the subclass to return the height of item in pixels.
Allow for two additional pixels for the list selection box.

Parameters

- **in item** The item whose height is returned.

Returns

- The height of the specified item in pixels.

See also:
- item_height(), item_width(), item_quick_height()  

Implemented in Fl_Check_Browser, and Fl_Browser.

### 12.14.4.20 item_last()

```cpp
virtual void * Fl_Browser_::item_last ( ) const [inline], [protected], [virtual]
```

This method must be provided by the subclass to return the last item in the list.

See also:
- item_first(), item_next(), item_last(), item_prev()  

Reimplemented in Fl_Browser.
12.14.4.21 item_next()

virtual void * Fl_Browser_::item_next (  
    void * item ) const [protected], [pure virtual]  
This method must be provided by the subclass to return the item in the list after item.

See also

    item_first(), item_next(), item_last(), item_prev()

Implemented in Fl_Check_Browser, and Fl_Browser.

12.14.4.22 item_prev()

virtual void * Fl_Browser_::item_prev (  
    void * item ) const [protected], [pure virtual]  
This method must be provided by the subclass to return the item in the list before item.

See also

    item_first(), item_next(), item_last(), item_prev()

Implemented in Fl_Check_Browser, and Fl_Browser.

12.14.4.23 item_quick_height()

int Fl_Browser_::item_quick_height (  
    void * item ) const [protected], [virtual]  
This method may be provided by the subclass to return the height of the item, in pixels.  
Allow for two additional pixels for the list selection box. This method differs from item_height in that it is only called for selection and scrolling operations. The default implementation calls item_height.

Parameters

| in  | item | The item whose height to return. |

Returns

The height, in pixels.

12.14.4.24 item_select()

void Fl_Browser_::item_select (  
    void * item,  
    int val = 1 ) [protected], [virtual]  
This method must be implemented by the subclass if it supports multiple selections; sets the selection state to val for the item.

Sets the selection state for item, where optional val is 1 (select, the default) or 0 (de-select).

Parameters

| in  | item | The item to be selected |
| in  | val  | The optional selection state; 1=select, 0=de-select. The default is to select the item (1). |

Reimplemented in Fl_Check_Browser, and Fl_Browser.

12.14.4.25 item_selected()

int Fl_Browser_::item_selected (  

This method must be implemented by the subclass if it supports multiple selections; returns the selection state for
item.
The method should return 1 if item is selected, or 0 otherwise.

Parameters

| in  | item | The item to test. |

Reimplemented in Fl_Check_Browser, and Fl_Browser.

12.14.4.26 item_swap()

virtual void Fl_Browser_::item_swap (void * a, void * b) [inline], [protected], [virtual]

This optional method should be provided by the subclass to efficiently swap browser items a and b, such as for sorting.

Parameters

| in  | a,b | The two items to be swapped. |

Reimplemented in Fl_Browser, and Fl_Check_Browser.

12.14.4.27 item_text()

virtual const char * Fl_Browser_::item_text (void * item) const [inline], [protected], [virtual]

This optional method returns a string (label) that may be used for sorting.

Parameters

| in  | item | The item whose label text is returned. |

Returns

The item's text label. (Can be NULL if blank)

Reimplemented in Fl_Browser, and Fl_Check_Browser.

12.14.4.28 item_width()

virtual int Fl_Browser_::item_width (void * item) const [protected], [pure virtual]

This method must be provided by the subclass to return the width of the item in pixels.
Allow for two additional pixels for the list selection box.

Parameters

| in  | item | The item whose width is returned. |
Returns
The width of the item in pixels.

Implemented in Fl_Check_Browser, and Fl_Browser.

12.14.4.29 leftedge()

int Fl_Browser_::leftedge ( ) const [protected]
This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.

Returns
The X position of the left edge of the list, in pixels.

See also
Fl_Browser_::bbox()

12.14.4.30 linespacing() [1/2]

int Fl_Browser_::linespacing ( ) const [inline]
Return the height of additional spacing between browser lines.

Returns
spacing height in pixel units.

12.14.4.31 linespacing() [2/2]

void Fl_Browser_::linespacing ( int pixels ) [inline]
Add some space between browser lines.

Parameters

| in   | pixels | number of additional pixels between lines |

12.14.4.32 new_list()

void Fl_Browser_::new_list ( ) [protected]
This method should be called when the list data is completely replaced or cleared. It informs the Fl_Browser_ widget that any cached information it has concerning the items is invalid. This method does not clear the list, it just handles the follow up bookkeeping after the list has been cleared.

12.14.4.33 position() [1/2]

int Fl_Browser_::position ( ) const [inline]

Deprecated "in 1.4.0 - use vposition() instead"

12.14.4.34 position() [2/2]

void Fl_Browser_::position ( int pos ) [inline]

Deprecated "in 1.4.0 - use vposition(pos) instead"
12.14.4.35 redraw_line()

```cpp
void Fl_Browser_::redraw_line ( void * item ) [protected]
```

This method should be called when the contents of item has changed, but not its height.

Parameters

- **in item** The item that needs to be redrawn.

See also

- `redraw_lines()`, `redraw_line()`

12.14.4.36 redraw_lines()

```cpp
void Fl_Browser_::redraw_lines ( ) [inline], [protected]
```

This method will cause the entire list to be redrawn.

See also

- `redraw_lines()`, `redraw_line()`

12.14.4.37 replacing()

```cpp
void Fl_Browser_::replacing ( void * a, void * b ) [protected]
```

This method should be used when item a is being replaced by item b. It allows the Fl_Browser_ to update its cache data as needed, schedules a redraw for the item being changed, and tries to maintain the selection. This method does not actually replace the item, but handles the follow up bookkeeping after the item has just been replaced.

Parameters

- **in a** Item being replaced
- **in b** Item to replace `a`

12.14.4.38 resize()

```cpp
void Fl_Browser_::resize ( int X, int Y, int W, int H ) [virtual]
```

Repositions and/or resizes the browser.

Parameters

- **in X,Y,W,H** The new position and size for the browser, in pixels.

Reimplemented from Fl_Widget.

12.14.4.39 scrollbar_left()

```cpp
void Fl_Browser_::scrollbar_left ( ) [inline]
```

Moves the vertical scrollbar to the lefthand side of the list.
For back compatibility.

12.14.40 scrollbar_right()
void Fl_Browser_::scrollbar_right ( ) [inline]
Moves the vertical scrollbar to the righthand side of the list.
For back compatibility.

12.14.41 scrollbar_size() [1/2]
int Fl_Browser_::scrollbar_size ( ) const [inline]
Gets the current size of the scrollbars' troughs, in pixels.
If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.

Returns
Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also
Fl::scrollbar_size(int)

12.14.42 scrollbar_size() [2/2]
void Fl_Browser_::scrollbar_size ( int newSize ) [inline]
Sets the pixel size of the scrollbars' troughs to newSize, in pixels.
Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.
Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare.
Setting newSize to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>newSize</th>
<th>Sets the scrollbar size in pixels.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()</td>
</tr>
</tbody>
</table>

See also
Fl::scrollbar_size()

12.14.43 scrollbar_width() [1/2]
int Fl_Browser_::scrollbar_width ( ) const [inline]
Returns the global value Fl::scrollbar_size().

Deprecated Use scrollbar_size() instead.
Todo This method should eventually be removed in 1.4+

12.14.44 scrollbar_width() [2/2]
void Fl_Browser_::scrollbar_width ( int width ) [inline]
Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

Deprecated Use scrollbar_size() instead.
Todo This method should eventually be removed in 1.4+
12.14.45 select()

```
int Fl_Browser_::select (  
    void * item,  
    int val = 1,  
    int docallbacks = 0 )  
```

Sets the selection state of `item` to `val`, and returns 1 if the state changed or 0 if it did not. If `docallbacks` is non-zero, select tries to call the callback function for the widget.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item whose selection state is to be changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>val</td>
<td>The new selection state (1=select, 0=de-select)</td>
</tr>
<tr>
<td>in</td>
<td>docallbacks</td>
<td>If non-zero, invokes widget callback if item changed. If 0, doesn't do callback (default).</td>
</tr>
</tbody>
</table>

Returns

1 if state was changed, 0 if not.

12.14.46 select_only()

```
int Fl_Browser_::select_only (  
    void * item,  
    int docallbacks = 0 )  
```

Selects `item` and returns 1 if the state changed or 0 if it did not. Any other items in the list are deselected.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item to select.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallbacks</td>
<td>If non-zero, invokes widget callback if item changed. If 0, doesn't do callback (default).</td>
</tr>
</tbody>
</table>

12.14.47 selection()

```
void * Fl_Browser_::selection ( ) const [inline], [protected]  
```

Returns the item currently selected, or NULL if there is no selection. For multiple selection browsers this call returns the currently focused item, even if it is not selected. To find all selected items, call `Fl_Multi_Browser::selected()` for every item in question.

12.14.48 sort()

```
void Fl_Browser_::sort (  
    int flags = 0 )  
```

Sort the items in the browser based on `flags`. `item_swap(void*, void*)` and `item_text(void*)` must be implemented for this call.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>flags</th>
<th>FL_SORT_ASCENDING – sort in ascending order</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FL_SORT_DESCENDING – sort in descending order</td>
<td>FL_SORT_CASEINSENSITIVE – add this to sort case-insensitively</td>
</tr>
<tr>
<td></td>
<td>Values other than the above will cause undefined behavior</td>
<td>Other flags may appear in the future.</td>
</tr>
</tbody>
</table>
12.14.49 swapping()

```cpp
void Fl_Browser_::swapping (  
  void ∗ a,  
  void ∗ b ) [protected]
```

This method should be used when two items \( a \) and \( b \) are being swapped.

It allows the `Fl_Browser_` to update its cache data as needed, schedules a redraw for the two items, and tries to maintain the current selection. This method does not actually swap items, but handles the follow up bookkeeping after items have been swapped.

**Parameters**

- **in \( a, b \)** Items being swapped.

12.14.50 textfont()

```cpp
Fl_Font Fl_Browser_::textfont ( ) const [inline]
```

Gets the default text font for the lines in the browser.

See also

- `textfont(), textsize(), textcolor()`

12.14.51 vposition() [1/2]

```cpp
int Fl_Browser_::vposition ( ) const [inline]
```

Gets the vertical scroll position of the list as a pixel position \( \text{pos} \).

The position returned is how many pixels of the list are scrolled off the top edge of the screen. Example: A position of '3' indicates the top 3 pixels of the list are scrolled off the top edge of the screen.

See also

- `position(), hposition()`

12.14.52 vposition() [2/2]

```cpp
void Fl_Browser_::vposition (  
  int ∗ pos )
```

Sets the vertical scroll position of the list to pixel position \( \text{pos} \).

The position is how many pixels of the list are scrolled off the top edge of the screen. Example: A position of '3' scrolls the top three pixels of the list off the top edge of the screen.

**Parameters**

- **in \( \text{pos} \)** The vertical position (in pixels) to scroll the browser to.

See also

- `vposition(), hposition()`

12.14.5 Member Data Documentation

12.14.5.1 hscrollbar

```cpp
Fl_Scrollbar Fl_Browser_::hscrollbar
```

Horizontal scrollbar.

Public, so that it can be accessed directly.
### 12.14.5.2 scrollbar

**Fl_Scrollbar Fl_Browser_::scrollbar**  
Vertical scrollbar.  
Public, so that it can be accessed directly.  
Use `scrollbar_left()` or `scrollbar_right()` to change what side the vertical scrollbar is drawn on.  
Use `scrollbar.align(int)` (see `Fl_Widget::align(Fl_Align)`) to change what side either of the scrollbars is drawn on.  
If the `FL_ALIGN_LEFT` bit is on, the vertical scrollbar is on the left. If the `FL_ALIGN_TOP` bit is on, the horizontal scrollbar is on the top. Note that only the alignment flags in scrollbar are considered. The flags in hscrollbar however are ignored.  
The documentation for this class was generated from the following files:

- `Fl_Browser_.H`
- `Fl_Browser_.cxx`

### 12.15 Fl_Button Class Reference

Buttons generate callbacks when they are clicked by the user.  
```
#include <Fl_Button.H>
```

Inheritance diagram for `Fl_Button`:

```
  Fl_Widget
   ^            ^
   |            |
  Fl_Button
   |            |
  Fl_Light_Button Fl_Radio_Button Fl_Repeat_Button Fl_Return_Button Fl_Shortcut_Button Fl_Toggle_Button
   |            |
  Fl_Check_Button Fl_Radio_Light_Button Fl_Round_Button Fl_Radio_Round_Button
```

#### Public Member Functions

- int `clear()`  
  *Same as `value(0)`.*
- uchar `compact()`  
  *Return true if buttons are rendered as compact buttons.*
- void `compact(uchar v)`  
  *Decide if buttons should be rendered in compact mode.*
- `Fl_Boxtype down_box()` const  
  *Returns the current down box type, which is drawn when `value()` is non-zero.*
- void `down_box(Fl_Boxtype b)`  
  *Sets the down box type.*
- `Fl_Color down_color()` const  
  *(for backwards compatibility)*
- void `down_color(unsigned c)`  
  *(for backwards compatibility)*
- `Fl_Button(int X, int Y, int W, int H, const char *L=0)`  
  *The constructor creates the button using the given position, size, and label.*
- int `handle(int)` **FL_OVERRIDE**  
  *Handles the specified event.*
- int `set()`  
  *Same as `value(1)`.*
- void `setonly()`  
  *Turns on this button and turns off all other radio buttons in the group (calling `value(1)` or `set()` does not do this).*
- int `shortcut()` const
Returns the current shortcut key for the button.

- void `shortcut` (const char ∗s)
  (for backwards compatibility)
- void `shortcut` (int s)
  Sets the shortcut key to s.
- char `value` () const
  Returns the current value of the button (0 or 1).
- int `value` (int v)
  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void `_clear_fullscreen` ()
- void `_set_fullscreen` ()
- void `activate` ()
  Activates the widget.
- unsigned int `active` () const
  Returns whether the widget is active.
- int `active_r` () const
  Returns whether the widget and all of its parents are active.
- Fl_Align `align` () const
  Gets the label alignment.
- void `align` (Fl_Align alignment)
  Sets the label alignment.
- long `argument` () const
  Gets the current user data (long) argument that is passed to the callback function.
- void `argument` (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗`as_gl_window` ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗`as_gl_window` () const
- virtual Fl_Group ∗`as_group` ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗`as_group` () const
- virtual Fl_Window ∗`as_window` ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗`as_window` () const
- void `bind_deimage` (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the inactive state.
- void `bind_deimage` (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void `bind_image` (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the active state.
- void `bind_image` (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype `box` () const
  Gets the box type of the widget.
- void `box` (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p `callback` () const
  Gets the current callback function for the widget.
• void callback (Fl_Callback *cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
  Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.
• void copy_label (const char *new_label)
  Sets the current label.
• void copy_tooltip (const char *text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image * deimage ()
Gets the image that is used as part of the widget label when in the inactive state.

- `const Fl_Image * deimage () const`
  Gets the image that is used as part of the widget label when in the inactive state.

- `void deimage (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the inactive state.

- `void deimage (Fl_Image *img)`
  Sets the image to use as part of the widget label when in the inactive state.

- `int deimage_bound () const`
  Returns whether the inactive image is managed by the widget.

- `void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with default arguments.

- `void do_callback (Fl_Widget *wgt, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.

- `void do_callback (Fl_Widget *wgt, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.

- `void draw_label (int, int, int, int, Fl_Align) const`
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- `int h () const`
  Gets the widget height.

- `virtual void hide ()`
  Makes a widget invisible.

- `Fl_Image * image ()`
  Gets the image that is used as part of the widget label when in the active state.

- `const Fl_Image * image () const`
  Gets the image that is used as part of the widget label when in the active state.

- `void image (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the active state.

- `void image (Fl_Image *img)`
  Sets the image to use as part of the widget label when in the active state.

- `int image_bound () const`
  Returns whether the image is managed by the widget.

- `int inside (const Fl_Widget *wgt) const`
  Checks if this widget is a child of `wgt`.

- `int is_label_copied () const`
  Returns whether the current label was assigned with `copy_label()`.

- `const char * label () const`
  Gets the current label text.

- `void label (const char *text)`
  Sets the current label pointer.

- `void label (Fl_LabelType a, const char *b)`
  Shortcut to set the label text and type in one call.

- `Fl_Color labelcolor () const`
  Gets the label color.

- `void labelcolor (Fl_Color c)`
  Sets the label color.

- `Fl_Font labelfont () const`
  Gets the font to use.

- `void labelfont (Fl_Font f)`
  Sets the font to use.

- `Fl_Fontsize labelsize () const`
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.

• Fl_Labeltype labeltype () const
  Gets the label type.

• void labeltype (Fl_Labeltype a)
  Sets the label type.

• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.

• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.

• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.

• unsigned int output () const
  Returns if a widget is used for output only.

• Fl_Group * parent () const
  Returns a pointer to the parent widget.

• void parent (Fl_Group * p)
  Internal use only - "for hacks only".

• void position (int X, int Y)
  Repositions the window or widget.

• void redraw ()
  Schedules the drawing of the widget.

• void redraw_label ()
  Schedules the drawing of the label.

• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.

• Fl_Color selection_color () const
  Gets the selection color.

• void selection_color (Fl_Color a)
  Sets the selection color.

• void set_active ()
  Marks the widget as active without sending events or changing focus.

• void set_changed ()
  Marks the value of the widget as changed.

• void set_output ()
  Sets a widget to output only.

• void set_visible ()
  Makes the widget visible.

• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.

• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.

• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

• virtual void show ()
  Makes a widget visible.

• void size (int W, int H)
  Changes the size of the widget.

• int take_focus ()
  Gives the widget the keyboard focus.

• unsigned int takesevents () const
Returns if the widget is able to take events.

- `int test_shortcut ()`
  Returns true if the widget's label contains the entered '&x' shortcut.

- `const char * tooltip () const`
  Gets the current tooltip text.

- `void tooltip (const char *text)`
  Sets the current tooltip text.

- `FL_Window * top_window () const`
  Returns a pointer to the top-level window for the widget.

- `FL_Window * top_window_offset (int &xoff, int &yoff) const`
  Finds the x/y offset of the current widget relative to the top-level window.

- `uchar type () const`
  Gets the widget type.

- `void type (uchar t)`
  Sets the widget type.

- `int use_accents_menu ()`
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- `void * user_data () const`
  Gets the user data for this widget.

- `void user_data (Fl_Callback_User_Data *v, bool auto_free)`
  Sets the user data for this widget.

- `void user_data (void *v)`
  Sets the user data for this widget.

- `unsigned int visible () const`
  Returns whether a widget is visible.

- `unsigned int visible_focus () const`
  Checks whether this widget has a visible focus.

- `void visible_focus (int v)`
  Modifies keyboard focus navigation.

- `int visible_r () const`
  Returns whether a widget and all its parents are visible.

- `int w () const`
  Gets the widget width.

- `Fl_When when () const`
  Returns the conditions under which the callback is called.

- `void when (uchar i)`
  Sets the flags used to decide when a callback is called.

- `FL_Window * window () const`
  Returns a pointer to the nearest parent window up the widget hierarchy.

- `int x () const`
  Gets the widget position in its window.

- `int y () const`
  Gets the widget position in its window.

- `virtual ~Fl_Widget ()`
  Destroys the widget.

### Protected Member Functions

- `void draw () FL_OVERRIDE`
  Draws the widget.

- `void simulate_key_action ()`
Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)
  Clears a flag in the flags mask.
- void **draw_backdrop** () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const
  Draws the widget box according its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void **draw_focus** () const
  Draws a focus rectangle around the widget.
- void **draw_focus** (Fl_Boxtype t, int X, int Y , int W, int H) const
  Draws a focus rectangle around the widget.
- void **draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void **draw_label** () const
  Draws the widget's label at the defined label position.
- void **draw_label** (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗ label=0L)
  Creates a widget at the given position and size.
- unsigned int **flags** () const
  Gets the widget flags mask.
- void h (int v)
  Internal use only.
- void **set_flag** (unsigned int c)
  Sets a flag in the flags mask.
- void w (int v)
  Internal use only.
- void x (int v)
  Internal use only.
- void y (int v)
  Internal use only.

Static Protected Member Functions

- static void **key_release_timeout** (void ∗)

Static Protected Attributes

- static Fl_Widget_Tracker ∗ **key_release_tracker** = 0

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void **default_callback** (Fl_Widget ∗ widget, void ∗ data)
  The default callback for all widgets that don't set a callback.
- static unsigned int **label_shortcut** (const char ∗ t)
  Returns the Unicode value of the '&'x' shortcut in a given text.
- static int **test_shortcut** (const char ∗ t, const bool require_alt=false)
  Returns true if the given text 't' contains the entered '&x' shortcut.
Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22, AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

12.15.1 Detailed Description

Buttons generate callbacks when they are clicked by the user. You control exactly when and how by changing the values for type(uchar) and when(uchar). Buttons can also generate callbacks in response to FL_SHORTCUT events. The button can either have an explicit shortcut(int s) value or a letter shortcut can be indicated in the label() with an '&' character before it. For the label shortcut it does not matter if Alt is held down, but if you have an input field in the same window, the user will have to hold down the Alt key so that the input field does not eat the event first as an FL_KEYBOARD event.

See also

Fl_Widget::shortcut_label(int)

For an Fl_Button object, the type() call returns one of:

- FL_NORMAL_BUTTON (0): value() remains unchanged after button press.
- FL_TOGGLE_BUTTON: value() is inverted after button press.
- FL_RADIO_BUTTON: value() is set to 1 after button press, and all other buttons in the current group with type() == FL_RADIO_BUTTON are set to zero.

For an Fl_Button object, the following when() values are useful, the default being FL_WHEN_RELEASE:

- 0: The callback is not done, instead changed() is turned on.
- FL_WHEN_RELEASE: The callback is done after the user successfully clicks the button, or when a shortcut is typed. The reason is FL_REASON_RELEASED.
- FL_WHEN_CHANGED: The callback is done each time the value() changes (when the user pushes and releases the button, and as the mouse is dragged around in and out of the button). The reason is set to FL_REASON_CHANGED
- FL_WHEN_NOT_CHANGED: The callback is done when the mouse button is released, but the value did not changed. The reason is set to FL_REASON_SELECTED

12.15.2 Constructor & Destructor Documentation

12.15.2.1 Fl_Button()

Fl_Button::Fl_Button ( int X, int Y, int W, int H, const char * L = 0 )

Generated by Doxygen
The constructor creates the button using the given position, size, and label. The default box type is box(FL_UP_BOX).

You can control how the button is drawn when ON by setting down_box(). The default is FL_NO_BOX (0) which will select an appropriate box type using the normal (OFF) box type by using fl_down(box()). Derived classes may handle this differently.

A button may request callbacks with when() FL_WHEN_CHANGED, FL_WHEN_NOT_CHANGED, and FL_WHEN_RELEASE, triggering the callback reasons FL_REASON_CHANGED, FL_REASON_SELECTED, and FL_REASON_DESELECTED.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X,Y,W,H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

### 12.15.3 Member Function Documentation

#### 12.15.3.1 clear()

```cpp
int Fl_Button::clear () [inline]
```

Same as value(0).

See also

`value(int v)`

#### 12.15.3.2 compact() [1/2]

```cpp
uchar Fl_Button::compact () [inline]
```

Return true if buttons are rendered as compact buttons.

Returns

0 if compact mode is off, 1 if it is on

See also

`compact(bool)`

#### 12.15.3.3 compact() [2/2]

```cpp
void Fl_Button::compact ( uchar v )
```

Decide if buttons should be rendered in compact mode.

---

**Figure 12.4 compact button keypad using GTK+ Scheme**
In compact mode, the button's surrounding border is altered to visually signal that multiple buttons are functionally linked together. To ensure the correct rendering of buttons in compact mode, all buttons must be part of the same group, positioned close to each other, and aligned with the edges of the group. Any button outlines not in contact with the parent group's outline will be displayed as separators.

Parameters

| in | v | switch compact mode on (1) or off (0) |

### 12.15.3.4 down_box() [1/2]

```
Fl_Boxtype Fl_Button::down_box ( ) const [inline]
```

Returns the current down box type, which is drawn when `value()` is non-zero.

Return values

```
Fl_Boxtype
```

### 12.15.3.5 down_box() [2/2]

```
void Fl_Button::down_box ( Fl_Boxtype b ) [inline]
```

Sets the down box type.

The default value of 0 causes FLTK to figure out the correct matching down version of `box()`.

Some derived classes (e.g. `Fl_Round_Button` and `Fl_Light_Button` use `down_box()` for special purposes. See docs of these classes.

Parameters

| in | b | down box type |

### 12.15.3.6 draw()

```
void Fl_Button::draw ( ) [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

---

*Generated by Doxygen*
FL_Widget *s = &Scrollbar;  // scrollbar is an embedded Fl_Scrollbar
s->draw();              // calls Fl_Scrollbar::draw()

Implements Fl_Widget.
Reimplemented in Fl_Light_Button, Fl_Return_Button, and Fl_Shortcut_Button.

12.15.3.7 handle()

int Fl_Button::handle (int event) [virtual]

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class.
This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in event | the kind of event received |

Return values

| 0 | if the event was not used or understood |
| 1 | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Widget.

Reimplemented in Fl_Light_Button, Fl_Repeat_Button, Fl_Return_Button, and Fl_Shortcut_Button.

12.15.3.8 set()

int Fl_Button::set ( ) [inline]

Same as value(1).

See also

value(int v)

12.15.3.9 shortcut() [1/2]

int Fl_Button::shortcut ( ) const [inline]

Returns the current shortcut key for the button.

Return values

| int | |

Generated by Doxygen
12.15.3.10  shortcut() [2/2]

void Fl_Button::shortcut ( int s ) [inline]

Sets the shortcut key to `s`.
Setting this overrides the use of `&` in the `label()`. The value is a bitwise OR of a key and a set of shift flags, for example: `FL_ALT | 'a'`, or `FL_ALT | (FL_F + 10)`, or just `'a'`. A value of 0 disables the shortcut.
The key can be any value returned by `Fl::event_key()`, but will usually be an ASCII letter. Use a lower-case letter unless you require the shift key to be held down.
The shift flags can be any set of values accepted by `Fl::event_state()`. If the bit is on, that shift key must be pushed. Meta, Alt, Ctrl, and Shift must be off if they are not in the shift flags (zero for the other bits indicates a "don't care" setting).

Parameters

| in | s | bitwise OR of key and shift flags |

12.15.3.11  value()

int Fl_Button::value ( int v )

Sets the current value of the button.
A non-zero value sets the button to 1 (ON), and zero sets it to 0 (OFF).

Parameters

| in | v | button value |

See also

`set()`, `clear()`

The documentation for this class was generated from the following files:

- Fl_Button.H
- Fl_Button.cxx

12.16  Fl_Cairo_State Class Reference

Contains all the necessary info on the current cairo context.

```cpp
#include <Fl_Cairo.H>
```

Public Member Functions

- bool autolink () const
  
  Gets the autolink option. See `Fl::cairo_autolink_context(bool)`
- void autolink (bool b)
  
  Sets the autolink option, only available with `-enable-cairoext`.
- cairo_t * cc () const
  
  Gets the current cairo context.
- void cc (cairo_t *c, bool own=true)
  
  Sets the current cairo context.
- void * gc () const
  
  Gets the last gc attached to a cc.
- void gc (void *c)
  
  Sets the gc `c` to keep track on.
• void *window () const
  
  Gets the last window attached to a cc.
• void window (void *w)
  
  Sets the window \textit{w} to keep track on.

### 12.16.1 Detailed Description

Contains all the necessary info on the current cairo context.

A private internal & unique corresponding object is created to permit cairo context state handling while keeping it opaque. For internal use only.

Note

Only available when configure has the \texttt{--enable-cairo} or \texttt{--enable-cairoext} option or one or both of the CMake options \texttt{OPTION\_CAIRO} or \texttt{OPTION\_CAIROEXT} is set (ON)

### 12.16.2 Member Function Documentation

#### 12.16.2.1 cc()

```cpp
void Fl_Cairo_State::cc (cairo_t *c, bool own = true) [inline]
```

Sets the current cairo context.

\texttt{own} == \texttt{true} (the default) indicates that the cairo context \texttt{c} will be deleted by FLTK internally when another cc is set later.

\texttt{own} == \texttt{false} indicates cc deletion is handled externally by the user program.

The documentation for this class was generated from the following files:

- \texttt{Fl\_Cairo.H}
- \texttt{Fl\_Cairo.cxx}

### 12.17 Fl\_Cairo\_Window Class Reference

This defines an FLTK window with Cairo support.

#include <Fl_Cairo_Window.H>

Inheritance diagram for Fl\_Cairo\_Window:

```
  Fl_Widget
  |     |
  |     | Fl_Group
  |     |
  |     | Fl_Window
  |     |
  |     | Fl_Double_Window
  |     |
  |     | Fl_Cairo_Window
```

**Public Types**

- typedef void(\*cairo_draw_cb) (Fl_Cairo_Window *self, cairo_t *def)

  The Cairo draw callback prototype you need to implement.

**Public Types inherited from Fl\_Window**

- typedef struct HICON__ = HICON
Public Member Functions

- **Fl_Cairo_Window** (int W, int H, const char *L=0)
- **Fl_Cairo_Window** (int X, int Y, int W, int H, const char *L=0)
- void **set_draw_cb** (cairo_draw_cb cb)
  
  You must provide a draw callback that implements your Cairo rendering.

Public Member Functions inherited from Fl_Double_Window

- **Fl_Double_Window** * as_double_window () FL_OVERRIDE
  Return non-null if this is an Fl_Overlay_Window object.
- **Fl_Double_Window** (int W, int H, const char *l=0)
  Creates a new Fl_Double_Window widget using the given position, size, and label (title) string.
- **Fl_Double_Window** (int X, int Y, int W, int H, const char *l=0)
  See Fl_Double_Window::Fl_Double_Window(int w, int h, const char *label = 0)
- void **flush** () FL_OVERRIDE
  Forces the window to be drawn, this window is also made current and calls draw().
- void **hide** () FL_OVERRIDE
  Makes a widget invisible.
- void **resize** (int, int, int, int) FL_OVERRIDE
  Changes the size or position of the widget.
- void **show** () FL_OVERRIDE
  Makes a widget visible.
- void **show** (int a, char **b)
  Same as Fl_Window::show(int a, char **b)
- **~Fl_Double_Window** ()
  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Window

- virtual class Fl_Overlay_Window * as_overlay_window ()
  Return non-null if this is an Fl_Overlay_Window object.
- **Fl_Window** const * as_window () const FL_OVERRIDE
- **Fl_Window** * as_window () FL_OVERRIDE
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int **border** () const
  Returns whether the window possesses a border.
- void **border** (int b)
  Sets whether or not the window manager border is around the window.
- void **clear_border** ()
  Fast inline function to turn the window manager border off.
- void **clear_modal_states** ()
  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.
- void **copy_label** (const char *a)
  Sets the window titlebar label to a copy of a character string.
- void **cursor** (const Fl_RGB_Image *, int, int)
  Changes the cursor for this window using the provided image as cursor's shape.
- void **cursor** (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  For back compatibility only.
- void **cursor** (Fl_Cursor)
  Changes the cursor for this window.
- int **decorated_h** () const
Returns the window height including any window title bar and any frame added by the window manager.

- **int decorated_w () const**
  
  Returns the window width including any frame added by the window manager.

- **void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)**
  
  For back compatibility only.

- **void default_cursor (Fl_Cursor)**
  
  Sets the default window cursor.

- **void draw_backdrop ()**
  
  Draw the background image if one is set and is aligned inside.

- **Fl_Window (int w, int h, const char ∗title=0)**
  
  Creates a window from the given width \(w\), height \(h\), and \(title\).

- **Fl_Window (int x, int y, int w, int h, const char ∗title=0)**
  
  Creates a window from the given position \((x, y)\), size \((w, h)\) and \(title\).

- **void free_position ()**
  
  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.

- **void fullscreen ()**
  
  Makes the window completely fill one or more screens, without any window manager border visible.

- **unsigned int fullscreen_active () const**
  
  Returns non zero if FULLSCREEN flag is set, 0 otherwise.

- **void fullscreen_off ()**
  
  Turns off any side effects of fullscreen()

- **void fullscreen_off (int X, int Y , int W, int H)**
  
  Turns off any side effects of fullscreen() and does resize(x,y,w,h).

- **void fullscreen_screens (int top, int bottom, int left, int right)**
  
  Sets which screens should be used when this window is in fullscreen mode.

- **int handle (int) FL_OVERRIDE**
  
  Handles the specified event.

- **void hide () FL_OVERRIDE**
  
  Removes the window from the screen.

- **void hotspot (const Fl_Widget &p, int offscreen=0)**
  
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

- **void hotspot (const Fl_Widget ∗, int offscreen=0)**
  
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

- **void hotspot (int x, int y, int offscreen=0)**
  
  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.

- **const void ∗icon () const**
  
  Gets the current icon window target dependent data.

- **void icon (const Fl_RGB_Image ∗)**
  
  Sets or resets a single window icon.

- **void icon (const void ∗ic)**
  
  Platform-specific method to set the window icon usable on Windows and X11 only.

- **void iconize ()**
  
  Iconifies the window.

- **const char ∗iconlabel () const**
  
  See void Fl_Window::iconlabel(const char ∗)

- **void iconlabel (const char ∗)**
  
  Sets the icon label.

- **void icons (const Fl_RGB_Image ∗[ ], int)**
Sets the window icons.

- void **icons** (HICON big_icon, HICON small_icon)
  
  Sets the window icons using HICON handles (Windows platform only).

- const char * **label** () const
  
  See void Fl_Window::label(const char*)

- void **label** (const char *)
  
  Sets the window title bar label.

- void **label** (const char *label, const char *iconlabel)
  
  Sets the icon label.

- void **make_current** ()
  
  Sets things up so that the drawing functions in `<FL/fl_draw.H>` will go into this window.

- void **maximize** ()
  
  Maximizes a top-level window to its current screen.

- unsigned int **maximize_active** () const
  
  Returns whether the window is currently maximized.

- unsigned int **menu_window** () const
  
  Returns true if this window is a menu window.

- unsigned int **modal** () const
  
  Returns true if this window is modal.

- unsigned int **non_modal** () const
  
  Returns true if this window is modal or non-modal.

- fl_uintptr_t **os_id** ()
  
  Returns a platform-specific identification of a shown window, or 0 if not shown.

- unsigned int **override** () const
  
  Returns non zero if OVERRIDE flag is set, 0 otherwise.

- void **resize** (int X, int Y, int W, int H) FL_OVERRIDE
  
  Changes the size and position of the window.

- int **screen_num** ()
  
  The number of the screen containing the mapped window.

- void **screen_num** (int screen_num)
  
  Set the number of the screen where to map the window.

- void **set_menu_window** ()
  
  Marks the window as a menu window.

- void **set_modal** ()
  
  A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program,
  and will also remain on top of the other windows (if the X window manager supports the "transient for" property).

- void **set_non_modal** ()
  
  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on
  top, but it has no effect on event delivery.

- void **set OVERRIDE** ()
  
  Activates the flags NOBORDER|OVERRIDE.

- void **set_tooltip_window** ()
  
  Marks the window as a tooltip window.

- const Fl_Image * **shape** ()
  
  Returns the image controlling the window shape or NULL.

- void **shape** (const Fl_Image &b)
  
  Set the window's shape with an Fl_Image.

- void **shape** (const Fl_Image *img)
  
  Assigns a non-rectangular shape to the window.

- void **show** () FL_OVERRIDE
Puts the window on the screen.

- **void** **show**(int argc, char **argv)
  
Puts the window on the screen with **show()** and parses command-line arguments.

- **int** **shown()**
  
  Returns non-zero if **show()** has been called (but not **hide()**).

- **void** **size_range**(int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)
  
  Sets the allowable range the user can resize this window to.

- **unsigned int** **tooltip_window()** const
  
  Returns true if this window is a tooltip window.

- **void** **un_maximize()**
  
  Returns a previously maximized top-level window to its previous size.

- **void** **wait_for_expose()**
  
  Waits for the window to be displayed after calling **show()**.

- **int** **x_root()** const
  
  Gets the x position of the window on the screen.

- **const char** **xclass()** const
  
  Returns the xclass for this window, or a default.

- **void** **xclass(const char *c)**
  
  Sets the xclass for this window.

- **int** **y_root()** const
  
  Gets the y position of the window on the screen.

- **virtual** ~**Fl_Window()**
  
  The destructor also deletes all the children.

**Public Member Functions inherited from Fl_Group**

- **Fl_Widget** *&&**_ddfdesign_kludge()**
  
  This is for forms compatibility only.

- **void** **add**(Fl_Widget &)
  
  The widget is removed from its current group (if any) and then added to the end of this group.

- **void** **add**(Fl_Widget *o)
  
  See void Fl_Group::add(Fl_Widget &w)

- **void** **add_resizable**(Fl_Widget &o)
  
  Adds a widget to the group and makes it the resizable widget.

- **Fl_Widget** *const* **array()** const
  
  Returns a pointer to the array of children.

- **Fl_Group** *const* **as_group()** const FL_OVERRIDE
  
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- **void** **begin()**
  
  Sets the current group so you can build the widget tree by just constructing the widgets.

- **Fl_Widget** *child**(int n) const
  
  Returns array()[n].

- **int** **children()** const
  
  Returns how many child widgets the group has.

- **void** **clear()**
  
  Deletes all child widgets from memory recursively.

- **unsigned int** **clip_children()**
  
  Returns the current clipping mode.

- **void** **clip_children**(int c)
  
  Controls whether the group widget clips the drawing of child widgets to its bounding box.
• void end ()
  Exactly the same as current(this->parent()).
• int find (const Fl_Widget &o) const
  See int Fl_Group::find(const Fl_Widget ∗w) const.
• int find (const Fl_Widget ∗) const
  Searches the child array for the widget and returns the index.
• Fl_Group (int, int, int, int, const char ∗=0)
  Creates a new Fl_Group widget using the given position, size, and label string.
• void focus (Fl_Widget ∗W)
• void forms_end ()
  This is for forms compatibility only.
• void init_sizes ()
  Resets the internal array of widget sizes and positions.
• void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.
• void insert (Fl_Widget &o, Fl_Widget ∗before)
  This does insert(w, find(before)).
• void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.
• void remove (Fl_Widget ∗o)
  Removes the widget o from the group.
• void remove (int index)
  Removes the widget at index from the group but does not delete it.
• Fl_Widget ∗ resizable () const
  Returns the group's resizable widget.
• void resizable (Fl_Widget &o)
• void resizable (Fl_Widget ∗o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
• virtual ~Fl_Group ()
  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  Activates the widget.
• unsigned int active () const
  Returns whether the widget is active.
• int active_r () const
  Returns whether the widget and all of its parents are active.
• Fl_Align align () const
  Gets the label alignment.
• void align (Fl_Align alignment)
  Sets the label alignment.
• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
• void argument (long v)

Generated by Doxygen
Sets the current user data (long) argument that is passed to the callback function.

- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const ∗ as_gl_window () const

- void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- Fl_Boxtype box () const
  Gets the box type of the widget.

- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.

- Fl_Callback_p callback () const
  Gets the current callback function for the widget.

- void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.

- void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.

- unsigned int changed () const
  Checks if the widget value changed since the last callback.

- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  Clears or sets the damage flags.

- void clear_output ()
  Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.

- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

- Fl_Color color () const
  Gets the background color of the widget.

- void color (Fl_Color bg)
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

- Fl_Color color2 () const
  For back compatibility only.

- void color2 (unsigned a)
For back compatibility only.

- **int contains (const Fl_Widget *w) const**
  Checks if w is a child of this widget.

- **void copy_label (const char *new_label)**
  Sets the current label.

- **void copy_tooltip (const char *text)**
  Sets the current tooltip text.

- **uchar damage () const**
  Returns non-zero if draw() needs to be called.

- **void damage (uchar c)**
  Sets the damage bits for the widget.

- **void damage (uchar c, int x, int y, int w, int h)**
  Sets the damage bits for an area inside the widget.

- **int damage_resize (int, int, int, int)**
  Internal use only.

- **void deactivate ()**
  Deactivates the widget.

- **Fl_Image *deimage ()**
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image *deimage () const**
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (Fl_Image &img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const**
  Returns whether the inactive image is managed by the widget.

- **void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with default arguments.

- **void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void draw_label (int, int, int, int, Fl_Align) const**
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h () const**
  Gets the widget height.

- **Fl_Image *image ()**
  Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image *image () const**
  Gets the image that is used as part of the widget label when in the active state.

- **void image (Fl_Image &img)**
  Sets the image to use as part of the widget label when in the active state.

- **void image (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const**
  Returns whether the image is managed by the widget.

- **int inside (const Fl_Widget *wgt) const**
  Checks if this widget is a child of wgt.

- **int is_label_copied () const**
  Returns whether the current label was assigned with copy_label().
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
Makes the widget visible.

- void **set_visible_focus** ()
  Enables keyboard focus navigation with this widget.

- int **shortcut_label** () const
  Returns whether the widget's label uses '&' to indicate shortcuts.

- void **shortcut_label** (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

- void **size** (int W, int H)
  Changes the size of the widget.

- int **take_focus** ()
  Gives the widget the keyboard focus.

- unsigned int **takesevents** () const
  Returns if the widget is able to take events.

- int **test_shortcut** ()
  Returns true if the widget's label contains the entered 'x' shortcut.

- const char ∗ **tooltip** () const
  Gets the current tooltip text.

- void **tooltip** (const char ∗text)
  Sets the current tooltip text.

- Fl_Window ∗ **top_window** () const
  Returns a pointer to the top-level window for the widget.

- Fl_Window ∗ **top_window_offset** (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar **type** () const
  Gets the widget type.

- void **type** (uchar t)
  Sets the widget type.

- int **use_accents_menu** ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void ∗ **user_data** () const
  Gets the user data for this widget.

- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.

- void ∗ **user_data** (void ∗v)
  Sets the user data for this widget.

- unsigned int **visible** () const
  Returns whether a widget is visible.

- unsigned int **visible_focus** () const
  Checks whether this widget has a visible focus.

- void **visible_focus** (int v)
  Modifies keyboard focus navigation.

- int **visible_r** () const
  Returns whether a widget and all its parents are visible.

- int **w** () const
  Gets the widget width.

- Fl_When **when** () const
  Returns the conditions under which the callback is called.

- void **when** (uchar i)
  Sets the flags used to decide when a callback is called.

- Fl_Window ∗ **window** () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.

Protected Member Functions

• void draw () FL_OVERRIDE
  Overloaded to provide Cairo callback support.

Protected Member Functions inherited from Fl_Window

• void default_size_range ()
  Protected method to calculate the default size range of a window.
• void draw () FL_OVERRIDE
  Draws the widget.
• int force_position () const
  Returns the internal state of the window's FORCE_POSITION flag.
• void force_position (int force)
  Sets an internal flag that tells FLTK and the window manager to honor position requests.
• void free_icons ()
  Deletes all icons previously attached to the window.
• int is_resizable ()
  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

• Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
• void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.
• void draw_children ()
  Draws all children of the group.
• void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.
• virtual int on_insert (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.
• virtual int on_move (int, int)
  Allow derived groups to act when a widget is moved within the group.
• virtual void on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.
• int * sizes ()
  Returns the internal array of widget sizes and positions.
• void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.
Protected Member Functions inherited from Fl_Widget

- void clear_flag(unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop() const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box() const
  Draws the widget box according its box style.
- void draw_box(Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus() const
  Draws a focus rectangle around the widget.
- void draw_focus(Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
- void draw_focus(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void draw_label() const
  Draws the widget's label at the defined label position.
- void draw_label(int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget(int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.
- unsigned int flags() const
  Gets the widget flags mask.
- void h(int v)
  Internal use only.
- void set_flag(unsigned int c)
  Sets a flag in the flags mask.
- void w(int v)
  Internal use only.
- void x(int v)
  Internal use only.
- void y(int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Window

- static Fl_Window *current()
  Returns the last window that was made current.
- static void default_callback(Fl_Window *, void *)
  Back compatibility: Sets the default callback v for win to call on close event.
- static void default_icon(const Fl_RGB_Image *)
  Sets a single default window icon.
- static void default_icons(const Fl_RGB_Image [], int)
  Sets the default window icons.
- static void default_icons(HICON big_icon, HICON small_icon)
  Sets the default window icons (Windows platform only).
- static const char *default_xclass()
Returns the default xclass.
• static void default_xclass (const char *)

Sets the default window xclass.
• static bool is_a_rescale ()

Returns true when a window is being rescaled.
• static char show_next_window_iconic ()

Returns the static flag whether the next window should be opened iconified.
• static void show_next_window_iconic (char stat)

Sets a static flag whether the next window should be opened iconified.

Static Public Member Functions inherited from Fl_Group

• static Fl_Group * current ()

Returns the currently active group.
• static void current (Fl_Group *g)

Sets the current group.

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget *widget, void *data)
The default callback for all widgets that don’t set a callback.
• static unsigned int label_shortcut (const char *t)

Returns the Unicode value of the ‘&x’ shortcut in a given text.
• static int test_shortcut (const char *t, const bool require_alt=false)

Returns true if the given text t contains the entered ‘&x’ shortcut.

Protected Types inherited from Fl_Widget

• enum {
INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
AUTO_DELETE_USER_DATA = 1<<23 ,
MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }

flags possible values enumeration.

Static Protected Attributes inherited from Fl_Window

• static Fl_Window * current_

Stores the last window that was made current.

12.17.1 Detailed Description
This defines an FLTK window with Cairo support.
This class overloads the virtual draw() method for you, so that the only thing you have to do is to provide your Cairo code. All Cairo context handling is achieved transparently.
The default coordinate system for Cairo drawing commands within Fl_Cairo_Window is FLTK’s coordinate system, where the x, y, w, h values are relative to the top/left corner of the Fl_Cairo_Window, as one would expect with regular FLTK drawing commands, e.g.: (0 x w-1), (0 y h-1). Example:
static void my_cairo_draw_cb(Fl_Cairo_Window *window, cairo_t *cr) {
    // Draw an "X"
    const double xmax = (window->w() - 1);
    const double ymax = (window->h() - 1);
    cairo_set_line_width(cr, 1.00); // line width for drawing
    cairo_set_source_rgb(cr, 1.0, 0.5, 0.0); // orange
    cairo_move_to(cr, 0.0, 0.0); cairo_line_to(cr, xmax, ymax); // draw diagonal "\"
    cairo_move_to(cr, 0.0, ymax); cairo_line_to(cr, xmax, 0.0); // draw diagonal "/"
    cairo_stroke(cr); // stroke the lines
}

The FLTK coordinate system differs from the default native Cairo coordinate system which uses normalized (0.0 ...
1.0) values for x and y, e.g.: (0 x 1.0), (0 y 1.0). So beware of this when copy/pasting Cairo example programs
that assume normalized values. If need be, you can revert to the Cairo coordinate system by simply calling
cairo_scale() with the widget's w() and h() values. **Example:**

```c
static void my_cairo_draw_cb(Fl_Cairo_Window *window, cairo_t *cr) {
    cairo_scale(cr, window->w(), window->h()); // use Cairo's default coordinate system
    \[..use 0.0 to 1.0 values from here on..\]
}
```

See also
- examples/cairo-draw-x.cxx
- test/cairo_test.cxx

**Note**

Class **Fl_Cairo_Window** requires the FLTK library to have been built with CMake option OPTION_CAIRO or
configure --enable-cairo.

You can alternatively define your custom Cairo FLTK window, and thus at least override the **draw()** method
to provide custom Cairo support. In this case you will probably use **Fl::cairo_make_current(Fl_Window*)** to
attach a context to your window. You should do this only when your window is the current window.

See also
- **Fl_Window::current()**

### 12.17.2 Member Function Documentation

#### 12.17.2.1 draw()

```c
void Fl_Cairo_Window::draw ()
```

Overloaded to provide Cairo callback support.

Implements **Fl_Widget**.

#### 12.17.2.2 set_draw_cb()

```c
void Fl_Cairo_Window::set_draw_cb ( cairo_draw_cb cb )
```

You must provide a draw callback that implements your Cairo rendering.

This method permits you to set your Cairo callback to **cb**.

The documentation for this class was generated from the following file:
- **Fl_Cairo_Window.H**

### 12.18 Fl_Callback_User_Data Class Reference

A class prototype that allows for additional data in callbacks.

```c
#include <Fl_Widget.H>
```
12.18.1 Detailed Description

A class prototype that allows for additional data in callbacks. Users can extend this class and pass it to widget callbacks. Widgets can take ownership of the callback data, deleting the data when the widget itself is deleted.

The destructor of this class is virtual, allowing for additional code to deallocate resources when the user data is deleted.

See also

FL_FUNCTION_CALLBACK_3, FL_METHOD_CALLBACK_1, FL_INLINE_CALLBACK_2

Fl_Widget::callback(Fl_Callback*, Fl_Callback_User_Data*, bool)
Fl_Widget::user_data(Fl_Callback_User_Data*, bool)

The documentation for this class was generated from the following file:

- Fl_Widget.H

12.19 Fl_Chart Class Reference

Fl_Chart displays simple charts.

#include <Fl_Chart.H>

Inheritance diagram for Fl_Chart:

```
Fl_Widget
  ↓
Fl_Chart
```

Public Member Functions

- void add (double val, const char *str=0, unsigned col=0)
  
  Adds the data value `val` with optional label `str` and color `col` to the chart.

- uchar autosize () const
  
  Gets whether the chart will automatically adjust the bounds of the chart.

- void autosize (uchar n)
  
  Sets whether the chart will automatically adjust the bounds of the chart.

- void bounds (double a, double b) const
  
  Gets the lower and upper bounds of the chart values.

- void bounds (double *a, double *b)
  
  Sets the lower and upper bounds of the chart values.

- void clear ()
  
  Removes all values from the chart.

- Fl_Chart (int X, int Y , int W, int H, const char *L=0)
  
  Create a new Fl_Chart widget using the given position, size and label string.

- void insert (int ind, double val, const char *str=0, unsigned col=0)
  
  Inserts a data value `val` at the given position `ind`.

- int maxsize () const
  
  Gets the maximum number of data values for a chart.

- void maxsize (int m)
  
  Sets the maximum number of data values for a chart.

- void replace (int ind, double val, const char *str=0, unsigned col=0)
  
  Replaces a data value `val` at the given position `ind`.

- int size () const
  
  Gets the number of data values in the chart.
Returns the number of data values in the chart.

- void size (int W, int H)
  Sets the widget size (width, height).

- Fl_Color textcolor () const
  Gets the chart's text color.

- void textcolor (Fl_Color n)
  Sets the chart's text color to n.

- Fl_Font textfont () const
  Gets the chart's text font.

- void textfont (Fl_Font s)
  Sets the chart's text font to s.

- Fl_Fontsize textsize () const
  Gets the chart's text size.

- void textsize (Fl_Fontsize s)
  Sets the chart's text size to s.

- ~Fl_Chart ()
  Destroys the Fl_Chart widget and all of its data.

Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.

- unsigned int active () const
  Returns whether the widget is active.

- int active_r () const
  Returns whether the widget and all of its parents are active.

- Fl_Align align () const
  Gets the label alignment.

- void align (Fl_Align alignment)
  Sets the label alignment.

- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.

- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const ∗ as_gl_window () const
- virtual Fl_Group ∗ as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- virtual Fl_Group const ∗ as_group () const
- virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const ∗ as_window () const
- void bind_deimage (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the active state.
void bind_image (int f)

Bind the image to the widget, so the widget will delete the image when it is no longer needed.

Fl_Boxtype box () const

Gets the box type of the widget.

void box (Fl_Boxtype new_box)

Sets the box type for the widget.

Fl_Callback_p callback () const

Gets the current callback function for the widget.

void callback (Fl_Callback *cb)

Sets the current callback function for the widget.

void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

Sets the current callback function and managed user data for the widget.

void callback (Fl_Callback *cb, void *p)

Sets the current callback function and data for the widget.

void callback (Fl_Callback0 *cb)

Sets the current callback function for the widget.

void callback (Fl_Callback1 *cb, long p=0)

Sets the current callback function for the widget.

unsigned int changed () const

Checks if the widget value changed since the last callback.

void clear_active ()

Marks the widget as inactive without sending events or changing focus.

void clear_changed ()

Marks the value of the widget as unchanged.

void clear_damage (uchar c=0)

Clears or sets the damage flags.

void clear_output ()

Sets a widget to accept input.

void clear_visible ()

Hides the widget.

void clear_visible_focus ()

Disables keyboard focus navigation with this widget.

Fl_Color color () const

Gets the background color of the widget.

void color (Fl_Color bg)

Sets the background color of the widget.

void color (Fl_Color bg, Fl_Color sel)

Sets the background and selection color of the widget.

Fl_Color color2 () const

For back compatibility only.

void color2 (unsigned a)

For back compatibility only.

int contains (const Fl_Widget *w) const

Checks if w is a child of this widget.

void copy_label (const char *new_label)

Sets the current label.

void copy_tooltip (const char *text)

Sets the current tooltip text.

uchar damage () const

Returns non-zero if draw() needs to be called.

void damage (uchar c)
Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image * deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image * deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *wgt, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *wgt, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual int handle (int event)
  Handles the specified event.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• FL_Color labelcolor () const
  Gets the label color.
• void labelcolor (FL_Color c)
  Sets the label color.
• FL_Font labelfont () const
  Gets the font to use.
• void labelfont (FL_Font f)
  Sets the font to use.
• FL_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (FL_Fontsize pix)
  Sets the font size in pixels.
• FL_Labeltype labeltype () const
  Gets the label type.
• void labeltype (FL_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• FL_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (FL_Group *p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
• FL_Color selection_color () const
  Gets the selection color.
• void selection_color (FL_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
Returns whether the widget's label uses '&' to indicate shortcuts.

- void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

- virtual void show ()
  Makes a widget visible.

- void size (int W, int H)
  Changes the size of the widget.

- int take_focus ()
  Gives the widget the keyboard focus.

- unsigned int takesevents () const
  Returns if the widget is able to take events.

- int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.

- const char * tooltip () const
  Gets the current tooltip text.

- void tooltip (const char *text)
  Sets the current tooltip text.

- Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.

- Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const
  Gets the widget type.

- void type (uchar t)
  Sets the widget type.

- int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void * user_data () const
  Gets the user data for this widget.

- void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

- void user_data (void *v)
  Sets the user data for this widget.

- unsigned int visible () const
  Returns whether a widget is visible.

- unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

- void visible_focus (int v)
  Modifies keyboard focus navigation.

- int visible_r () const
  Returns whether a widget and all its parents are visible.

- int w () const
  Gets the widget width.

- Fl_When when () const
  Returns the conditions under which the callback is called.

- void when (uchar i)
  Sets the flags used to decide when a callback is called.

- Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

- int x () const
  Gets the widget position in its window.
• int y () const
  
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  
  Destroys the widget.

Protected Member Functions

• void draw () FL_OVERRIDE
  
  Draws the Fl_Chart widget.

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  
  Clears a flag in the flags mask.

• void draw_backdrop () const
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• void draw_box () const
  
  Draws the widget box according its box style.

• void draw_box (Fl_Boxtype t, Fl_Color c) const
  
  Draws a box of type t, of color c at the widget's position and size.

• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type t, of color c at the position X,Y and size W,H.

• void draw_focus () const
  
  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h) const
  
  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.

• void draw_label () const
  
  Draws the widget's label at the defined label position.

• void draw_label (int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.

• Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  
  Creates a widget at the given position and size.

• unsigned int flags () const
  
  Gets the widget flags mask.

• void h (int v)
  
  Internal use only.

• void set_flag (unsigned int c)
  
  Sets a flag in the flags mask.

• void w (int v)
  
  Internal use only.

• void x (int v)
  
  Internal use only.

• void y (int v)
  
  Internal use only.
Static Protected Member Functions

- static void draw_barchart (int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[], double min, double max, int autosize, int maxnumb, FL_Color textcolor)
  
  Draws a bar chart.

- static void draw_horbarchart (int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[], double min, double max, int autosize, int maxnumb, FL_Color textcolor)

  Draws a horizontal bar chart.

- static void draw_linechart (int type, int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[], double min, double max, int autosize, int maxnumb, FL_Color textcolor)

  Draws a line chart.

- static void draw_piechart (int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[], int special, FL_Color textcolor)

  Draws a pie chart.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPYLABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPYTOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }

flags possible values enumeration.

12.19.1 Detailed Description

Fl_Chart displays simple charts.
It is provided for Forms compatibility.
Todo  Refactor Fl_Chart::type() information.

The type of an Fl_Chart object can be set using type(uchar t) to:

<table>
<thead>
<tr>
<th>Chart Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_BAR_CHART</td>
<td>Each sample value is drawn as a vertical bar.</td>
</tr>
<tr>
<td>FL_FILLED_CHART</td>
<td>The chart is filled from the bottom of the graph to the sample values.</td>
</tr>
<tr>
<td>FL_HORBAR_CHART</td>
<td>Each sample value is drawn as a horizontal bar.</td>
</tr>
<tr>
<td>FL_LINE_CHART</td>
<td>The chart is drawn as a polyline with vertices at each sample value.</td>
</tr>
<tr>
<td>FL_PIE_CHART</td>
<td>A pie chart is drawn with each sample value being drawn as a proportionate slice in the circle.</td>
</tr>
<tr>
<td>FL_SPECIALPIE_CHART</td>
<td>Like FL_PIE_CHART, but the first slice is separated from the pie.</td>
</tr>
<tr>
<td>FL_SPIKE_CHART</td>
<td>Each sample value is drawn as a vertical line.</td>
</tr>
</tbody>
</table>

12.19.2 Constructor & Destructor Documentation

12.19.2.1 Fl_Chart()

Fl_Chart::Fl_Chart (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char ∗ L = 0 )  

Create a new Fl_Chart widget using the given position, size and label string. The default boxstyle is FL_NO_BOX.

Parameters

| in   | X, Y, W, H | position and size of the widget |
| in   | L          | widget label, default is no label |

12.19.3 Member Function Documentation

12.19.3.1 add()

void Fl_Chart::add (  
    double val,  
    const char ∗ str = 0,  
    unsigned col = 0 )  

Adds the data value val with optional label str and color col to the chart.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
<th>data value</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>str</td>
<td>optional data label</td>
</tr>
<tr>
<td>in</td>
<td>col</td>
<td>optional data color</td>
</tr>
</tbody>
</table>

### 12.19.3.2 autosize(1/2)

**uchar Fl_Chart::autosize() const [inline]**

Gets whether the chart will automatically adjust the bounds of the chart.

**Returns**

non-zero if auto-sizing is enabled and zero if disabled.

### 12.19.3.3 autosize(2/2)

**void Fl_Chart::autosize(uchar n) [inline]**

Sets whether the chart will automatically adjust the bounds of the chart.

**Parameters**

|   in  | n | non-zero to enable automatic resizing, zero to disable |

### 12.19.3.4 bounds(1/2)

**void Fl_Chart::bounds(double *a, double *b) const [inline]**

Gets the lower and upper bounds of the chart values.

**Parameters**

| out   | a,b | are set to lower, upper |

### 12.19.3.5 bounds(2/2)

**void Fl_Chart::bounds(double a, double b)**

Sets the lower and upper bounds of the chart values.

**Parameters**

| in    | a,b | are used to set lower, upper |

### 12.19.3.6 draw()

**void Fl_Chart::draw() [protected], [virtual]**

Draws the Fl.Chart widget.

Implements Fl_Widget.
12.19.3.7 **draw_barchart()**

```c
void Fl_Chart::draw_barchart (  
    int x,  
    int y,  
    int w,  
    int h,  
    int numb,  
    FL_CHART_ENTRY entries[],  
    double min,  
    double max,  
    int autosize,  
    int maxnumb,  
    Fl_Color textcolor ) [static], [protected]
```

Draws a bar chart.

- **x, y, w, h** is the bounding box, **entries** the array of **numb** entries, and **min** and **max** the boundaries.

### Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x, y, w, h</td>
<td>Widget position and size</td>
</tr>
<tr>
<td>in</td>
<td>numb</td>
<td>Number of values</td>
</tr>
<tr>
<td>in</td>
<td>entries</td>
<td>Array of values</td>
</tr>
<tr>
<td>in</td>
<td>min</td>
<td>Lower boundary</td>
</tr>
<tr>
<td>in</td>
<td>max</td>
<td>Upper boundary</td>
</tr>
<tr>
<td>in</td>
<td>autosize</td>
<td>Whether the chart autosizes</td>
</tr>
<tr>
<td>in</td>
<td>maxnumb</td>
<td>Maximal number of entries</td>
</tr>
<tr>
<td>in</td>
<td>textcolor</td>
<td>Text color</td>
</tr>
</tbody>
</table>

12.19.3.8 **draw_horbarchart()**

```c
void Fl_Chart::draw_horbarchart (  
    int x,  
    int y,  
    int w,  
    int h,  
    int numb,  
    FL_CHART_ENTRY entries[],  
    double min,  
    double max,  
    int autosize,  
    int maxnumb,  
    Fl_Color textcolor ) [static], [protected]
```

Draws a horizontal bar chart.

- **x, y, w, h** is the bounding box, **entries** the array of **numb** entries, and **min** and **max** the boundaries.

### Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x, y, w, h</td>
<td>Widget position and size</td>
</tr>
<tr>
<td>in</td>
<td>numb</td>
<td>Number of values</td>
</tr>
<tr>
<td>in</td>
<td>entries</td>
<td>Array of values</td>
</tr>
<tr>
<td>in</td>
<td>min</td>
<td>Lower boundary</td>
</tr>
<tr>
<td>in</td>
<td>max</td>
<td>Upper boundary</td>
</tr>
<tr>
<td>in</td>
<td>autosize</td>
<td>Whether the chart autosizes</td>
</tr>
<tr>
<td>in</td>
<td>maxnumb</td>
<td>Maximal number of entries</td>
</tr>
<tr>
<td>in</td>
<td>textcolor</td>
<td>Text color</td>
</tr>
</tbody>
</table>
12.19.3.9  draw_linechart()

void Fl_Chart::draw_linechart (  
    int type,  
    int x,  
    int y,  
    int w,  
    int h,  
    int numb,  
    FL_CHART_ENTRY entries[],  
    double min,  
    double max,  
    int autosize,  
    int maxnumb,  
    Fl_Color textcolor ) [static], [protected]

Draws a line chart. 
*x, y, w, h* is the bounding box, *entries* the array of *numb* entries, and *min* and *max* the boundaries.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>type</th>
<th>Chart type</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y,w,h</td>
<td>Widget position and size</td>
</tr>
<tr>
<td>in</td>
<td>numb</td>
<td>Number of values</td>
</tr>
<tr>
<td>in</td>
<td>entries</td>
<td>Array of values</td>
</tr>
<tr>
<td>in</td>
<td>min</td>
<td>Lower boundary</td>
</tr>
<tr>
<td>in</td>
<td>max</td>
<td>Upper boundary</td>
</tr>
<tr>
<td>in</td>
<td>autosize</td>
<td>Whether the chart autosizes</td>
</tr>
<tr>
<td>in</td>
<td>maxnumb</td>
<td>Maximal number of entries</td>
</tr>
<tr>
<td>in</td>
<td>textcolor</td>
<td>Text color</td>
</tr>
</tbody>
</table>

12.19.3.10  draw_piechart()

void Fl_Chart::draw_piechart (  
    int x,  
    int y,  
    int w,  
    int h,  
    int numb,  
    FL_CHART_ENTRY entries[],  
    int special,  
    Fl_Color textcolor ) [static], [protected]

Draws a pie chart.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x,y,w,h</th>
<th>bounding box</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>numb</td>
<td>number of chart entries</td>
</tr>
<tr>
<td>in</td>
<td>entries</td>
<td>array of chart entries</td>
</tr>
<tr>
<td>in</td>
<td>special</td>
<td>special (?)</td>
</tr>
<tr>
<td>in</td>
<td>textcolor</td>
<td>text color</td>
</tr>
</tbody>
</table>

12.19.3.11  insert()

void Fl_Chart::insert (  
    int ind,

double val,
const char * str = 0,
unsigned col = 0)

Inserts a data value \texttt{val} at the given position \texttt{ind}.
Position 1 is the first data value.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>\texttt{ind}</th>
<th>insertion position</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>\texttt{val}</td>
<td>data value</td>
</tr>
<tr>
<td>in</td>
<td>\texttt{str}</td>
<td>optional data label</td>
</tr>
<tr>
<td>in</td>
<td>\texttt{col}</td>
<td>optional data color</td>
</tr>
</tbody>
</table>

12.19.3.12 \texttt{maxsize()}

```cpp
void Fl_Chart::maxsize (int m)
```

Sets the maximum number of data values for a chart.
If you do not call this method then the chart will be allowed to grow to any size depending on available memory.

**Parameters**

| in  | \texttt{m}  | maximum number of data values allowed. |

12.19.3.13 \texttt{replace()}

```cpp
void Fl_Chart::replace (int ind,
     double val,
     const char * str = 0,
     unsigned col = 0)
```

Replaces a data value \texttt{val} at the given position \texttt{ind}.
Position 1 is the first data value.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>\texttt{ind}</th>
<th>insertion position</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>\texttt{val}</td>
<td>data value</td>
</tr>
<tr>
<td>in</td>
<td>\texttt{str}</td>
<td>optional data label</td>
</tr>
<tr>
<td>in</td>
<td>\texttt{col}</td>
<td>optional data color</td>
</tr>
</tbody>
</table>

12.19.3.14 \texttt{size()}

```cpp
void Fl_Chart::size (int \_W, int \_H) [inline]
```

Sets the widget size (width, height).
This is the same as calling \texttt{Fl_Widget::size(int \_W, int \_H)};

**Parameters**

| in  | \texttt{W,H} | new width and height of the widget |

The documentation for this class was generated from the following files:
12.20 FL_CHART_ENTRY Struct Reference

For internal use only.

#include <Fl_Chart.H>

Public Attributes

- unsigned col
  
  For internal use only.

- char str [FL_CHART_LABEL_MAX+1]
  
  For internal use only.

- float val
  
  For internal use only.

12.20.1 Detailed Description

For internal use only.

The documentation for this struct was generated from the following file:

- Fl_Chart.H

12.21 Fl_Check_Browser Class Reference

The Fl_Check_Browser widget displays a scrolling list of text lines that may be selected and/or checked by the user.

#include <Fl_Check_Browser.H>

Inheritance diagram for Fl_Check_Browser:

```
Fl_Widget
    |
    |
    v
Fl_Group
    |
    |
    v
Fl_Browser_
    |
    |
    v
Fl_Check_Browser
```

Public Member Functions

- int add (char *s)
  
  Add a new unchecked line to the end of the browser.

- int add (char *s, int b)
  
  Add a new line to the end of the browser.

- int add (const char *s)
  
  See int Fl_Check_Browser::add(char *s)

- int add (const char *s, int b)
  
  See int Fl_Check_Browser::add(char *s)

- void check_all ()
  
  Sets all the items checked.

- void check_none ()
Sets all the items unchecked.

- **int checked (int item) const**
  
  Gets the current status of item item.

- **void checked (int item, int b)**
  
  Sets the check status of item item to b.

- **void clear ()**
  
  Remove every item from the browser.

- **Fl_Check_Browser (int x, int y, int w, int h, const char ∗l=0)**
  
  The constructor makes an empty browser.

- **void ∗ item_at (int index) const FL_OVERRIDE**
  
  This method must be provided by the subclass to return the item for the specified index.

- **void item_swap (int ia, int ib)**
  
  This optional method should be provided by the subclass to efficiently swap browser items a and b, such as for sorting.

- **int nchecked () const**
  
  Returns how many items are currently checked.

- **int nitems () const**
  
  Returns how many lines are in the browser.

- **int remove (int item)**
  
  Remove line n and make the browser one line shorter.

- **void set_checked (int item)**
  
  Equivalent to Fl_Check_Browser::checked(item, 1).

- **char ∗ text (int item) const**
  
  Return a pointer to an internal buffer holding item item's text.

- **int value () const**
  
  Returns the index of the currently selected item.

- **∼Fl_Check_Browser ()**
  
  The destructor deletes all list items and destroys the browser.

---

**Public Member Functions inherited from Fl_Browser_**

- **int deselect (int docallbacks=0)**
  
  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.

- **void display (void ∗item)**
  
  Displays the item, scrolling the list as necessary.

- **uchar has_scrollbar () const**
  
  Returns the current scrollbar mode, see Fl_Browser_::has_scrollbar(uchar)

- **void has_scrollbar (uchar mode)**
  
  Sets whether the widget should have scrollbars or not (default Fl_Browser_::BOTH).

- **int hposition () const**
  
  Gets the horizontal scroll position of the list as a pixel position pos.

- **void hposition (int)**
  
  Sets the horizontal scroll position of the list to pixel position pos.

- **int linespacing () const**
  
  Return the height of additional spacing between browser lines.

- **void linespacing (int pixels)**
  
  Add some space between browser lines.

- **int position () const**
  
  

- **void position (int pos)**
  
  

- **void position (int x, int y)**
  
  

- **void resize (int X, int Y, int W, int H) FL_OVERRIDE**
  
  

Repositions and/or resizes the browser.

- void scrollbar_left()
  Moves the vertical scrollbar to the lefthand side of the list.
- void scrollbar_right()
  Moves the vertical scrollbar to the righthand side of the list.
- int scrollbar_size() const
  Gets the current size of the scrollbars’ troughs, in pixels.
- void scrollbar_size(int newSize)
  Sets the pixel size of the scrollbars’ troughs to newSize, in pixels.
- int scrollbar_width() const
  Returns the global value Fl::scrollbar_size().
- void scrollbar_width(int width)
  Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.
- int select(void *item, int val=1, int docallbacks=0)
  Sets the selection state of item to val, and returns 1 if the state changed or 0 if it did not.
- int select_only(void *item, int docallbacks=0)
  Selects item and returns 1 if the state changed or 0 if it did not.
- void sort(int flags=0)
  Sorts the items in the browser based on flags.
- Fl_Color textcolor() const
  Gets the default text color for the lines in the browser.
- void textcolor(Fl_Color col)
  Sets the default text color for the lines in the browser to color col.
- Fl_Font textfont() const
  Gets the default text font for the lines in the browser.
- void textfont(Fl_Font font)
  Sets the default text font for the lines in the browser to font.
- Fl_Fontsize textsize() const
  Gets the default text size (in pixels) for the lines in the browser.
- void textsize(Fl_Fontsize newSize)
  Sets the default text size (in pixels) for the lines in the browser to size.
- int vposition() const
  Gets the vertical scroll position of the list as a pixel position pos.
- void vposition(int pos)
  Sets the vertical scroll position of the list to pixel position pos.

Public Member Functions inherited from Fl_Group

- Fl_Widget * & _ddfdesign_kludge()
  This is for forms compatibility only.
- void add(Fl_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.
- void add(Fl_Widget &o)
  See void Fl_Group::add(Fl_Widget &w)
- void add_resizable(Fl_Widget &o)
  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget * const * array() const
  Returns a pointer to the array of children.
- Fl_Group * const * as_group() const FL_OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• void **begin ()**
  
  *Sets the current group so you can build the widget tree by just constructing the widgets.*

• Fl_Widget *child (int n) const
  
  Returns array()[n].

• int **children () const**
  
  *Returns how many child widgets the group has.*

• void **clear ()**
  
  *Deletes all child widgets from memory recursively.*

• unsigned int **clip_children ()**
  
  *Returns the current clipping mode.*

• void **clip_children (int c)**
  
  *Controls whether the group widget clips the drawing of child widgets to its bounding box.*

• virtual int **delete_child (int n)**
  
  *Removes the widget at index from the group and deletes it.*

• void **end ()**
  
  *Exactly the same as current(this->parent()).*

• int **find (const Fl_Widget &o) const**
  
  See int Fl_Group::find(const Fl_Widget *w) const.

• int **find (const Fl_Widget *) const**
  
  *Searches the child array for the widget and returns the index.*

• Fl_Group (int, int, int, int, const char *)=0)
  
  *Creates a new Fl_Group widget using the given position, size, and label string.*

• void **focus (Fl_Widget &W)**

• void **forms_end ()**
  
  *This is for forms compatibility only.*

• int **handle (int) FL_OVERRIDE**
  
  *Handles the specified event.*

• void **init_sizes ()**
  
  *Resets the internal array of widget sizes and positions.*

• void **insert (Fl_Widget &, int i)**
  
  *The widget is removed from its current group (if any) and then inserted into this group.*

• void **insert (Fl_Widget &o, Fl_Widget *before)**
  
  *This does insert(w, find(before)).*

• void **remove (Fl_Widget &)**
  
  *Removes a widget from the group but does not delete it.*

• void **remove (Fl_Widget &o)**
  
  *Removes the widget o from the group.*

• void **remove (int index)**
  
  *Removes the widget at index from the group but does not delete it.*

• Fl_Widget *resizable () const
  
  *Returns the group's resizable widget.*

• void **resizable (Fl_Widget &o)**
  
  *Sets the group's resizable widget.*

• void **resizable (Fl_Widget *o)**
  
  *The resizable widget defines both the resizing box and the resizing behavior of the group and its children.*

• void **resize (int, int, int) FL_OVERRIDE**
  
  *Resizes the Fl_Group widget and all of its children.*

• virtual ~Fl_Group ()
  
  *The destructor also deletes all the children.*
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
- unsigned int active () const
- int active_r () const
- Fl_Align align () const
- void align (Fl_Align alignment)
- long argument () const
- void argument (long v)
- virtual class Fl_Gl_Window * as_gl_window ()
- virtual class Fl_Gl_Window const * as_gl_window () const
- virtual Fl_Window * as_window ()
- virtual Fl_Window const * as_window () const
- void bind_deimage (Fl_I mage * img)
- void bind_deimage (int f)
- void bind_image (Fl_I mage * img)
- void bind_image (int f)
- Fl_Boxtype box () const
- void box (Fl_Boxtype new_box)
- Fl_Callback_p callback () const
- void callback (Fl_Callback * cb)
- void callback (Fl_Callback * cb, Fl_Callback_User_Data * p, bool auto_free)
- void callback (Fl_Callback * cb, void * p)
- void callback (Fl_Callback0 * cb)
- unsigned int changed () const
- void clear_active ()
Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  Clears or sets the damage flags.

- void clear_output ()
  Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.

- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

- Fl_Color color () const
  Gets the background color of the widget.

- void color (Fl_Color bg)
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

- Fl_Color color2 () const
  For back compatibility only.

- void color2 (unsigned a)
  For back compatibility only.

- int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.

- void copy_label (const char *new_label)
  Sets the current label.

- void copy_tooltip (const char *text)
  Sets the current tooltip text.

- uchar damage () const
  Returns non-zero if draw() needs to be called.

- void damage (uchar c)
  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- int damage_resize (int, int, int, int)
  Internal use only.

- void deactivate ()
  Deactivates the widget.

- Fl_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

- const Fl_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

- void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

- void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_Callback_Reas reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reas reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void **do_callback** (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void **draw_label** (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int **h** () const
  Gets the widget height.
• virtual void **hide** ()
  Makes a widget invisible.
• Fl_Image * **image** ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image * **image** () const
  Gets the image that is used as part of the widget label when in the active state.
• void **image** (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void **image** (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int **image_bound** () const
  Returns whether the image is managed by the widget.
• int **inside** (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int **is_label_copied** () const
  Returns whether the current label was assigned with copy_label().
• const char * **label** () const
  Gets the current label text.
• void **label** (const char *text)
  Sets the current label pointer.
• void **label** (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color **labelcolor** () const
  Gets the label color.
• void **labelcolor** (Fl_Color c)
  Sets the label color.
• Fl_Font **labelfont** () const
  Gets the font to use.
• void **labelfont** (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize **labelsize** () const
  Gets the font size in pixels.
• void **labelsize** (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype **labeltype** () const
  Gets the label type.
• void **labeltype** (Fl_Labeltype a)
  Sets the label type.
• void **measure_label** (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool **needs_keyboard** () const
  Returns whether this widget needs a keyboard.
• void **needs_keyboard** (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int **output** () const
Returns if a widget is used for output only.

- **Fl_Group * parent () const**
  
  Returns a pointer to the parent widget.

- **void parent (Fl_Group *p)**
  
  Internal use only - "for hacks only".

- **void position (int X, int Y)**
  
  Repositions the window or widget.

- **void redraw ()**
  
  Schedules the drawing of the widget.

- **void redraw_label ()**
  
  Schedules the drawing of the label.

- **Fl_Color selection_color () const**
  
  Gets the selection color.

- **void selection_color (Fl_Color a)**
  
  Sets the selection color.

- **void set_active ()**
  
  Marks the widget as active without sending events or changing focus.

- **void set_changed ()**
  
  Marks the value of the widget as changed.

- **void set_output ()**
  
  Sets a widget to output only.

- **void set_visible ()**
  
  Makes the widget visible.

- **void set_visible_focus ()**
  
  Enables keyboard focus navigation with this widget.

- **int shortcut_label () const**
  
  Returns whether the widget's label uses '&' to indicate shortcuts.

- **void shortcut_label (int value)**
  
  Sets whether the widget's label uses '&' to indicate shortcuts.

- **virtual void show ()**
  
  Makes a widget visible.

- **void size (int W, int H)**
  
  Changes the size of the widget.

- **int take_focus ()**
  
  Gives the widget the keyboard focus.

- **unsigned int takesevents () const**
  
  Returns if the widget is able to take events.

- **int test_shortcut ()**
  
  Returns true if the widget's label contains the entered 'x' shortcut.

- **const char * tooltip () const**
  
  Gets the current tooltip text.

- **void tooltip (const char *text)**
  
  Sets the current tooltip text.

- **Fl_Window * top_window () const**
  
  Returns a pointer to the top-level window for the widget.

- **Fl_Window * top_window_offset (int &xoff, int &yoff) const**
  
  Finds the x/y offset of the current widget relative to the top-level window.

- **uchar type () const**
  
  Gets the widget type.

- **void type (uchar t)**
  
  Sets the widget type.
• int **use_accents_menu**
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void **user_data**() const

  Gets the user data for this widget.

• void **user_data**(FL_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.

• void **user_data**(void *v)

  Sets the user data for this widget.

• unsigned int **visible**() const

  Returns whether a widget is visible.

• unsigned int **visible_focus**() const

  Checks whether this widget has a visible focus.

• void **visible_focus**(int v)

  Modifies keyboard focus navigation.

• int **visible_r**() const

  Returns whether a widget and all its parents are visible.

• int **w**() const

  Gets the widget width.

• FL_When **when**() const

  Returns the conditions under which the callback is called.

• void **when**(uchar i)

  Sets the flags used to decide when a callback is called.

• FL_Window **window**() const

  Returns a pointer to the nearest parent window up the widget hierarchy.

• int **x**() const

  Gets the widget position in its window.

• int **y**() const

  Gets the widget position in its window.

• virtual ~**Fl_Widget**()

  Destroys the widget.

Protected Member Functions

• int **handle**(int) FL_OVERRIDE

  Handles the event within the normal widget bounding box.

• void **item_draw**(void *, int, int, int, int) const FL_OVERRIDE

  This method must be provided by the subclass to draw the item in the area indicated by X, Y, W, H.

• void **item_first**(void *) const FL_OVERRIDE

  This method must be provided by the subclass to return the first item in the list.

• int **item_height**(void *) const FL_OVERRIDE

  This method must be provided by the subclass to return the height of item in pixels.

• void **item_next**(void *) const FL_OVERRIDE

  This method must be provided by the subclass to return the item in the list after item.

• void **item_prev**(void *) const FL_OVERRIDE

  This method must be provided by the subclass to return the item in the list before item.

• void **item_select**(void *, int) FL_OVERRIDE

  This method must be implemented by the subclass if it supports multiple selections; sets the selection state to val for the item.

• int **item_selected**(void *) const FL_OVERRIDE

  This method must be implemented by the subclass if it supports multiple selections; returns the selection state for item.

• const char **item_text**(void *) const FL_OVERRIDE
This optional method returns a string (label) that may be used for sorting.

- **int item_width (void *) const FL_OVERRIDE**

  This method must be provided by the subclass to return the width of the item in pixels.

**Protected Member Functions inherited from Fl_Browser_**

- **void bbox (int &X, int &Y, int &W, int &H) const**

  Returns the bounding box for the interior of the list's display window, inside the scrollbars.

- **void deleting (void *item)**

  This method should be used when item is being deleted from the list.

- **int displayed (void *item) const**

  Returns non-zero if item has been scrolled to a position where it is being displayed.

- **void draw () FL_OVERRIDE**

  Draws the list within the normal widget bounding box.

- **void *find_item (int ypos)**

  This method returns the item under mouse y position ypos.

- **Fl_Browser_ (int X, int Y, int W, int H, const char *L=0)**

  The constructor makes an empty browser.

- **virtual int full_height () const**

  This method may be provided by the subclass to indicate the full height of the item list, in pixels.

- **virtual int full_width () const**

  This method may be provided by the subclass to indicate the full width of the item list, in pixels.

- **virtual int incr_height () const**

  This method may be provided to return the average height of all items to be used for scrolling.

- **void inserting (void *a, void *b)**

  This method should be used when an item is in the process of being inserted into the list.

- **virtual void *item_last () const**

  This method must be provided by the subclass to return the last item in the list.

- **virtual int item_quick_height (void *item) const**

  This method may be provided by the subclass to return the height of the item, in pixels.

- **int leftedge () const**

  This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.

- **void new_list ()**

  This method should be called when the list data is completely replaced or cleared.

- **void redraw_line (void *item)**

  This method should be called when the contents of item has changed, but not its height.

- **void redraw_lines ()**

  This method will cause the entire list to be redrawn.

- **void replacing (void *a, void *b)**

  This method should be used when item a is being replaced by item b.

- **void *selection () const**

  Returns the item currently selected, or NULL if there is no selection.

- **void swapping (void *a, void *b)**

  This method should be used when two items a and b are being swapped.

- **void *top () const**

  Returns the item that appears at the top of the list.
Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
- void draw () FL_OVERRIDE
  Draws the widget.
- void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.
- void draw_children ()
  Draws all children of the group.
- void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)
  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()
  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void draw_label () const
  Draws the widget's label at the defined label position.
- void draw_label (int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.
- unsigned int flags () const
  Gets the widget flags mask.
- void h (int v)
  Internal use only.
- void set_flag (unsigned int c)
Sets a flag in the flags mask.
• void w (int v)
  Internal use only.
• void x (int v)
  Internal use only.
• void y (int v)
  Internal use only.

Additional Inherited Members

Public Types inherited from Fl_Browser_
• enum {
  HORIZONTAL = 1, VERTICAL = 2, BOTH = 3, ALWAYS_ON = 4,
  HORIZONTAL_ALWAYS = 5, VERTICAL_ALWAYS = 6, BOTH_ALWAYS = 7 }
  Values for has_scrollbar() for has_scrollbar() for has_scrollbar().

Static Public Member Functions inherited from Fl_Group
• static Fl_Group * current ()
  Returns the currently active group.
• static void current (Fl_Group *g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget
• static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don’t set a callback.
• static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the ‘&x’ shortcut in a given text.
• static int test_shortcut (const char *t, const bool require_alt=false)
  Returns true if the given text t contains the entered ‘&x’ shortcut.

Public Attributes inherited from Fl_Browser_
• Fl_Scrollbar hscrollbar
  Horizontal scrollbar.
• Fl_Scrollbar scrollbar
  Vertical scrollbar.

Protected Types inherited from Fl_Widget
• enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE杨OCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_FACENTS =
  MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31}
  flags possible values enumeration.
12.21 Fl_Check_Browser Class Reference

12.21.1 Detailed Description

The Fl_Check_Browser widget displays a scrolling list of text lines that may be selected and/or checked by the user.

12.21.2 Member Function Documentation

12.21.2.1 add() [1/2]

```cpp
int Fl_Check_Browser::add ( char * s )
```

Add a new unchecked line to the end of the browser.

See also

```cpp
add(char *s, int b)
```

12.21.2.2 add() [2/2]

```cpp
int Fl_Check_Browser::add ( char * s, int b )
```

Add a new line to the end of the browser.

The text is copied using the strdup() function. It may also be NULL to make a blank line. It can set the item checked if b is not 0.

12.21.2.3 handle()

```cpp
int Fl_Check_Browser::handle ( int event ) [protected], [virtual]
```

Handles the event within the normal widget bounding box.

Parameters

- in event The event to process.

Returns

1 if event was processed, 0 if not.

Reimplemented from Fl_Browser_.

12.21.2.4 item_at()

```cpp
void * Fl_Check_Browser::item_at ( int index ) const [virtual]
```

This method must be provided by the subclass to return the item for the specified index.

Parameters

- in index The index of the item to be returned

Returns

The item at the specified index.

Reimplemented from Fl_Browser_.

12.21.2.5 item_draw()

```cpp
void Fl_Check_Browser::item_draw ( void * item,
```

Generated by Doxygen
**12.21.2.6 item_first()**

```cpp
void *Fl_Check_Browser::item_first () const [protected], [virtual]
```

This method must be provided by the subclass to return the first item in the list.

See also

- item_first(), item_next(), item_last(), item_prev()

Implements Fl_Browser_.

**12.21.2.7 item_height()**

```cpp
int Fl_Check_Browser::item_height (void *item) const [protected], [virtual]
```

This method must be provided by the subclass to return the height of item in pixels. Allow for two additional pixels for the list selection box.

**Parameters**

- **in item** - The item whose height is returned.

Returns

The height of the specified item in pixels.

See also

- item_height(), item_width(), item_quick_height()

Implements Fl_Browser_.

**12.21.2.8 item_next()**

```cpp
void *Fl_Check_Browser::item_next (void *item) const [protected], [virtual]
```

This method must be provided by the subclass to return the item in the list after item.

See also

- item_first(), item_next(), item_last(), item_prev()

Implements Fl_Browser_.

**12.21.2.9 item_prev()**

```cpp
void *Fl_Check_Browser::item_prev (void *item) const [protected], [virtual]
```

This method must be provided by the subclass to return the item in the list before item.

See also

- item_first(), item_next(), item_last(), item_prev()

Implements Fl_Browser_.

```

```
12.21.2.10  item_select()

```cpp
void Fl_Check_Browser::item_select (  
    void * item,  
    int val ) [protected], [virtual]
```

This method must be implemented by the subclass if it supports multiple selections; sets the selection state to \texttt{val}
for the \texttt{item}.

Sets the selection state for \texttt{item}, where optional \texttt{val} is 1 (select, the default) or 0 (de-select).

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>\texttt{item}</th>
<th>The item to be selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>\texttt{val}</td>
<td>The optional selection state; 1=select, 0=de-select. The default is to select the item (1).</td>
</tr>
</tbody>
</table>

Reimplemented from \texttt{Fl_Browser}.

12.21.2.11  item_selected()

```cpp
int Fl_Check_Browser::item_selected (  
    void * item ) const [protected], [virtual]
```

This method must be implemented by the subclass if it supports multiple selections; returns the selection state for
\texttt{item}.

The method should return 1 if \texttt{item} is selected, or 0 otherwise.

**Parameters**

| in  | \texttt{item} | The item to test. |

Reimplemented from \texttt{Fl_Browser}.

12.21.2.12  item_swap()

```cpp
void Fl_Check_Browser::item_swap (  
    void * a,  
    void * b ) [virtual]
```

This optional method should be provided by the subclass to efficiently swap browser items \texttt{a} and \texttt{b}, such as for
sorting.

**Parameters**

| in  | \texttt{a,b} | The two items to be swapped. |

Reimplemented from \texttt{Fl_Browser}.

12.21.2.13  item_text()

```cpp
const char * Fl_Check_Browser::item_text (  
    void * item ) const [protected], [virtual]
```

This optional method returns a string (label) that may be used for sorting.

**Parameters**

| in  | \texttt{item} | The item whose label text is returned. |
Returns

The item's text label. (Can be NULL if blank)

Reimplemented from Fl_Browser_.

### 12.21.2.14 item_width()

```c
int Fl_Check_Browser::item_width ( 
    void ∗ item ) const [protected], [virtual]
```

This method must be provided by the subclass to return the width of the item in pixels. Allow for two additional pixels for the list selection box.

**Parameters**

- `item` The item whose width is returned.

**Returns**

The width of the item in pixels.

Implements Fl_Browser_.

### 12.21.2.15 nitems()

```c
int Fl_Check_Browser::nitems ( ) const [inline]
```

Returns how many lines are in the browser. The last line number is equal to this.

### 12.21.2.16 remove()

```c
int Fl_Check_Browser::remove ( 
    int item )
```

Remove line n and make the browser one line shorter. Returns the number of lines left in the browser.

The documentation for this class was generated from the following files:

- `Fl_Check_Browser.H`
- `Fl_Check_Browser.cxx`

### 12.22 FL_Check_Button Class Reference

A button with a "checkmark" to show its status.

Inheritance diagram for FL_Check_Button:

```
Fl_Widget
   |
   v
Fl_Button
   |
   v
Fl_Light_Button
   |
   v
FL_Check_Button
```

**Public Member Functions**

- `FL_Check_Button (int X, int Y, int W, int H, const char ∗L=0)`

  Creates a new FL_Check_Button widget using the given position, size, and label string.
Public Member Functions inherited from Fl_Light_Button

- Fl_Light_Button (int x, int y, int w, int h, const char *l=0)
  
  Creates a new Fl_Light_Button widget using the given position, size, and label string.

- int handle (int) FL_OVERRIDE
  
  Handles the specified event.

Public Member Functions inherited from Fl_Button

- int clear ()
  
  Same as value(0).

- uchar compact ()
  
  Return true if buttons are rendered as compact buttons.

- void compact (uchar v)
  
  Decide if buttons should be rendered in compact mode.

- Fl_Boxtype down_box () const
  
  Returns the current down box type, which is drawn when value() is non-zero.

- void down_box (Fl_Boxtype b)
  
  Sets the down box type.

- Fl_Color down_color () const
  
  (for backwards compatibility)

- void down_color (unsigned c)
  
  (for backwards compatibility)

- Fl_Button (int X, int Y, int W, int H, const char *L=0)
  
  The constructor creates the button using the given position, size, and label.

- int set ()
  
  Same as value(1).

- void setonly ()
  
  Turns on this button and turns off all other radio buttons in the group. (calling value(1) or set() does not do this).

- int shortcut () const
  
  Returns the current shortcut key for the button.

- void shortcut (const char *s)
  
  (for backwards compatibility)

- void shortcut (int s)
  
  Sets the shortcut key to s.

- char value () const
  
  Returns the current value of the button (0 or 1).

- int value (int v)
  
  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()

- void _set_fullscreen ()

- void activate ()
  
  Activates the widget.

- unsigned int active () const
  
  Returns whether the widget is active.

- int active_r () const
  
  Returns whether the widget and all of its parents are active.

- Fl_Align align () const
  
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.

- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.

- virtual class Fl_Gl_Window * as_gl_window()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const * as_gl_window() const

- virtual Fl_Group * as_group()
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- virtual Fl_Group const * as_group() const

- virtual Fl_Window * as_window()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const * as_window() const

- void bind_deimage (Fl_Image * img)
  Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image (Fl_Image * img)
  Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- Fl_Boxtype box () const
  Gets the box type of the widget.

- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.

- Fl_Callback_p callback () const
  Gets the current callback function for the widget.

- void callback (Fl_Callback * cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback * cb, Fl_Callback_User_Data * p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback * cb, void * p)
  Sets the current callback function and data for the widget.

- void callback (Fl_Callback0 * cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback1 * cb, long p=0)
  Sets the current callback function for the widget.

- unsigned int changed () const
  Checks if the widget value changed since the last callback.

- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  Clears or sets the damage flags.

- void clear_output ()
  Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

• Fl_Color color () const
  Gets the background color of the widget.

• void color (Fl_Color bg)
  Sets the background color of the widget.

• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

• Fl_Color color2 () const
  For back compatibility only.

• void color2 (unsigned a)
  For back compatibility only.

• int contains (const Fl_Widget ∗ w) const
  Checks if w is a child of this widget.

• void copy_label (const char ∗ new_label)
  Sets the current label.

• void copy_tooltip (const char ∗ text)
  Sets the current tooltip text.

• uchar damage () const
  Returns non-zero if draw() needs to be called.

• void damage (uchar c)
  Sets the damage bits for the widget.

• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

• int damage_resize (int, int, int, int)
  Internal use only.

• void deactivate ()
  Deactivates the widget.

• Fl_Image ∗ deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

• const Fl_Image ∗ deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

• void deimage (Fl_Image & img)
  Sets the image to use as part of the widget label when in the inactive state.

• void deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.

• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

• void do_callback (Fl_Callback _Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void do_callback (Fl_Callback _Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• int h () const
  Gets the widget height.

• virtual void hide ()
  Makes a widget invisible.

• Fl_Image ∗ image ()
Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image ∗ image () const**
  
  Gets the image that is used as part of the widget label when in the active state.

- **void image (Fl_Image &img)**
  
  Sets the image to use as part of the widget label when in the active state.

- **void image (Fl_Image ∗img)**
  
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const**
  
  Returns whether the image is managed by the widget.

- **int inside (const Fl_Widget ∗wgt) const**
  
  Checks if this widget is a child of `wgt`.

- **int is_label_copied () const**
  
  Returns whether the current label was assigned with `copy_label()`.

- **const char ∗ label () const**
  
  Gets the current label text.

- **void label (const char ∗text)**
  
  Sets the current label pointer.

- **void label (Fl_Labeltype a, const char ∗b)**
  
  Shortcut to set the label text and type in one call.

- **Fl_Color labelcolor () const**
  
  Gets the label color.

- **void labelcolor (Fl_Color c)**
  
  Sets the label color.

- **Fl_Font labelfont () const**
  
  Gets the font to use.

- **void labelfont (Fl_Font f)**
  
  Sets the font to use.

- **Fl_Fontsize labelsize () const**
  
  Gets the font size in pixels.

- **void labelsize (Fl_Fontsize pix)**
  
  Sets the font size in pixels.

- **Fl_Labeltype labeltype () const**
  
  Gets the label type.

- **void labeltype (Fl_Labeltype a)**
  
  Sets the label type.

- **void measure_label (int &ww, int &hh) const**
  
  Sets width `ww` and height `hh` accordingly with the label size.

- **bool needs_keyboard () const**
  
  Returns whether this widget needs a keyboard.

- **void needs_keyboard (bool needs)**
  
  Sets whether this widget needs a keyboard.

- **unsigned int output () const**
  
  Returns if a widget is used for output only.

- **Fl_Group ∗ parent () const**
  
  Returns a pointer to the parent widget.

- **void parent (Fl_Group ∗p)**
  
  Internal use only - “for hacks only”.

- **void position (int X, int Y)**
  
  Repositions the window or widget.

- **void redraw ()**
  
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int testShortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.
• const char * tooltip () const
  Gets the current tooltip text.
• void tooltip (const char *text)
  Sets the current tooltip text.
• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
Sets the user data for this widget.

- unsigned int visible() const
  Returns whether a widget is visible.
- unsigned int visible_focus() const
  Checks whether this widget has a visible focus.
- void visible_focus(int v)
  Modifies keyboard focus navigation.
- int visible_r() const
  Returns whether a widget and all its parents are visible.
- int w() const
  Gets the widget width.
- Fl_When when() const
  Returns the conditions under which the callback is called.
- void when(uchar i)
  Sets the flags used to decide when a callback is called.
- Fl_Window* window() const
  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x() const
  Gets the widget position in its window.
- int y() const
  Gets the widget position in its window.
- virtual ~Fl_Widget()
  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback(Fl_Widget* widget, void* data)
  The default callback for all widgets that don't set a callback.
- static unsigned int labelShortcut(const char* t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int testShortcut(const char*, const bool require_alt=false)
  Returns true if the given text t contains the entered "&x" shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22, AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
Protected Member Functions inherited from Fl_Light_Button

• void draw () FL_OVERRIDE
  Draws the widget.

Protected Member Functions inherited from Fl_Button

• void simulate_key_action ()

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const
  Draws the widget box according its box style.
• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
• void draw_focus () const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
• void draw_label () const
  Draws the widget's label at the defined label position.
• void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
• Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.
• unsigned int flags () const
  Gets the widget flags mask.
• void h (int v)
  Internal use only.
• void set_flag (unsigned int c)
  Sets a flag in the flags mask.
• void w (int v)
  Internal use only.
• void x (int v)
  Internal use only.
• void y (int v)
  Internal use only.

Static Protected Member Functions inherited from Fl_Button

• static void key_release_timeout (void ∗)

Static Protected Attributes inherited from Fl_Button

• static Fl_Widget_Tracker ∗ key_release_tracker = 0

Generated by Doxygen
12.22.1 Detailed Description

A button with a "checkmark" to show its status.

![Fl_Check_Button]

Figure 12.7 Fl_Check_Button

Buttons generate callbacks when they are clicked by the user. You control exactly when and how by changing the values for type() and when().

The Fl_Check_Button subclass displays its "ON" state by showing a "checkmark" rather than drawing itself pushed in.

12.22.2 Constructor & Destructor Documentation

12.22.2.1 Fl_Check_Button()

Fl_Check_Button::Fl_Check_Button (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char ∗ L = 0 )

Creates a new Fl_Check_Button widget using the given position, size, and label string. The default box type is FL_NO_BOX, which draws the label w/o a box right of the checkmark. The selection_color() sets the color of the checkmark. Default is FL_FOREGROUND_COLOR (usually black). You can use down_box() to change the box type of the checkmark. Default is FL_DOWN_BOX.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X,Y,W,H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Check_Button.H
- Fl_Check_Button.cxx

12.23 Fl_Choice Class Reference

A button that is used to pop up a menu.

```cpp
#include <Fl_Choice.H>
```

Inheritance diagram for Fl_Choice:

```
Fl_Widget
    ↓
Fl_Menu_
    ↓
Fl_Choice
    ↓
Fl_Scheme_Choice
```

Public Member Functions

- Fl_Choice (int X, int Y, int W, int H, const char ∗L=0)
Create a new Fl_Choice widget using the given position, size and label string.

- **int handle (int) FL_OVERRIDE**
  Handles the specified event.
- **int value () const**
  Gets the index of the last item chosen by the user.
- **int value (const Fl_Menu_Item *v)**
  Sets the currently selected value using a pointer to menu item.
- **int value (int v)**
  Sets the currently selected value using the index into the menu item array.

### Public Member Functions inherited from Fl_Menu_

- **int add (const char *)**
  This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with |' separating the menu items, and tab separating the menu item names from an optional shortcut string.
  ```c
  int add (const char *, int shortcut, Fl_Callback *, void *, int=0)
  Adds a new menu item.
  ```
  ```c
  int add (const char *, a, const char *, b, Fl_Callback *, c, void *, d=0, int e=0)
  See int Fl_Menu_::add(const char* label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)
  ```
- **void clear ()**
  Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.
- **int clear_submenu (int index)**
  Clears the specified submenu pointed to by index of all menu items.
- **void copy (const Fl_Menu_Item *, void *, user_data=0)**
  Sets the menu array pointer with a copy of m that will be automatically deleted.
- **Fl_Boxtype down_box () const**
  This box type is used to surround the currently-selected items in the menus.
- **void down_box (Fl_Boxtype b)**
  Sets the box type used to surround the currently-selected items in the menus.
- **Fl_Color down_color () const**
  For back compatibility, same as selection_color()
- **void down_color (unsigned c)**
  For back compatibility, same as selection_color()
- **int find_index (const char *) const**
  Find the menu item index for a given menu pathname, such as "Edit/Copy".
- **int find_index (const Fl_Menu_Item *, item) const**
  Find the index into the menu array for a given item.
- **int find_index (Fl_Callback *, cb) const**
  Find the index into the menu array for a given callback cb.
- **const Fl_Menu_Item * find_item (const char *, name) const**
  Find the menu item for the given menu pathname, such as "Edit/Copy".
- **const Fl_Menu_Item * find_item (Fl_Callback *)**
  Find the menu item for the given callback cb.
- **const Fl_Menu_Item * find_item_with_argument (long v)**
  Find the menu item for the given user argument v.
- **const Fl_Menu_Item * find_item_with_user_data (void *)**
  Find the menu item for the given user data v.
- **Fl_Menu_ (int, int, int, int, const char * = 0)**
  Creates a new Fl_Menu_ widget using the given position, size, and label string.
- **void global ()**
  Make the shortcuts for this menu work no matter what window has the focus when you type it.
• int insert (int index, const char *, int shortcut, Fl_Callback *, void *=0, int e=0)
  Inserts a new menu item at the specified index position.
• int insert (int index, const char *a, const char *b, Fl_Callback *c, void *d=0, int e=0)
  See int Fl_Menu_::insert(const char* label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)
• int item_pathname (char *name, int namelen, const Fl_Menu_Item *finditem=0) const
  Get the menu 'pathname' for the specified menuitem.
• const Fl_Menu_Item * menu () const
  Returns a pointer to the array of Fl_Menu_Items.
• void menu (const Fl_Menu_Item *m)
  Sets the menu array pointer directly.
• Fl_Boxtype menu_box () const
  Get the box type for the menu popup windows.
• void menu_box (Fl_Boxtype b)
  Set the box type for the menu popup windows.
• const Fl_Menu_Item * menu_end ()
  Finishes menu modifications and returns menu().
• int mode (int i) const
  Gets the flags of item i.
• void mode (int i, int fl)
  Sets the flags of item i.
• const Fl_Menu_Item * mvalue () const
  Returns a pointer to the last menu item that was picked.
• const Fl_Menu_Item * picked (const Fl_Menu_Item *)
  When user picks a menu item, call this.
• const Fl_Menu_Item * prev_mvalue () const
  Returns a pointer to the menu item that was picked before the current one was picked.
• void remove (int)
  Deletes item i from the menu.
• void replace (int, const char *)
  Changes the text of item i.
• void setonly (Fl_Menu_Item *item)
  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.
• void shortcut (int i, int s)
  Changes the shortcut of item i to s.
• int size () const
  This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.
• void size (int W, int H)
• const Fl_Menu_Item * testShortcut ()
  Returns the menu item with the entered shortcut (key value).
• const char * text () const
  Returns the title of the last item chosen.
• const char * text (int i) const
  Returns the title of item i.
• Fl_Color textcolor () const
  Get the current color of menu item labels.
• void texture (Fl_Color c)
  Sets the current color of menu item labels.
• Fl_Font textfont () const
  Gets the current font of menu item labels.
• void textfont (Fl_Font c)
  Sets the current font of menu item labels.
- \texttt{Fl_Fontsize \texttt{textsize}()} const
  
  \textit{Gets the font size of menu item labels.}

- void \texttt{textsize(Fl_Fontsize\ c)}
  
  \textit{Sets the font size of menu item labels.}

- int \texttt{value()} const
  
  \textit{Returns the index into \texttt{menu()} of the last item chosen by the user.}

- int \texttt{value(const Fl_Menu.Item *\ i)}
  
  \textit{The value is the index into \texttt{menu()} of the last item chosen by the user.}

- int \texttt{value(int\ i)}
  
  \textit{The value is the index into \texttt{menu()} of the last item chosen by the user.}

**Public Member Functions inherited from \texttt{Fl_Widget}**

- void \_\texttt{clear_fullscreen()}\()
  
  Activates the widget.

- void \_\texttt{set_fullscreen()}\()
  
  
  \textit{Activates the widget.}

- unsigned int \texttt{active()} const
  
  \textit{Returns whether the widget is active.}

- int \texttt{active_r()} const
  
  \textit{Returns whether the widget and all of its parents are active.}

- \texttt{Fl.Align align()} const
  
  \textit{Gets the label alignment.}

- void \texttt{align(Fl.Align\ \texttt{alignment})}
  
  \textit{Sets the label alignment.}

- long \texttt{argument()} const
  
  \textit{Gets the current user data (long) argument that is passed to the callback function.}

- void \texttt{argument(long\ \texttt{v})}
  
  \textit{Sets the current user data (long) argument that is passed to the callback function.}

- virtual \texttt{class Fl.Gl_Window\ \ast as_gl_window()}
  
  Returns a \texttt{Fl.Gl_Window} pointer if this widget is an \texttt{Fl.Gl_Window}.

- virtual \texttt{class Fl.Gl_Window} const \texttt{as_gl_window()}\ const
  
  Returns a \texttt{Fl.Gl_Window} pointer if this widget is an \texttt{Fl.Gl_Window}.

- virtual \texttt{Fl.Group\ \ast as_group()}\()
  
  Returns a \texttt{Fl.Group} pointer if this widget is an \texttt{Fl.Group}.

- virtual \texttt{Fl.Group} const \texttt{as_group()}\ const
  
  Returns a \texttt{Fl.Group} pointer if this widget is an \texttt{Fl.Group}.

- virtual \texttt{Fl.Window\ \ast as_window()}\()
  
  Returns a \texttt{Fl.Window} pointer if this widget is an \texttt{Fl.Window}.

- virtual \texttt{Fl.Window} const \texttt{as_window()}\ const
  
  Sets the image to use as part of the widget label when in the inactive state.

- void \texttt{bind_deimage(Fl.Image\ \ast img)}
  
  \textit{Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.}

- void \texttt{bind_deimage(int\ f)}
  
  \textit{Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.}

- void \texttt{bind_image(Fl.Image\ \ast\ img)}
  
  \textit{Sets the image to use as part of the widget label when in the active state.}

- void \texttt{bind_image(int\ f)}
  
  \textit{Sets the image to use as part of the widget label when in the active state.}

- \texttt{Fl.Boxtype box()}\()
  
  \textit{Gets the box type of the widget.}

- void \texttt{box(Fl.Boxtype\ new_box)}
  
  \textit{Sets the box type for the widget.}

- \texttt{Fl.Callback\ p\ \texttt{callback}()}\ const
  
  \textit{Generated by Doxygen}
Gets the current callback function for the widget.

- **void callback (Fl_Callback ∗cb)**
  - Sets the current callback function for the widget.

- **void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)**
  - Sets the current callback function and managed user data for the widget.

- **void callback (Fl_Callback ∗cb, void ∗p)**
  - Sets the current callback function and data for the widget.

- **void callback (Fl_Callback0 ∗cb)**
  - Sets the current callback function for the widget.

- **void callback (Fl_Callback1 ∗cb, long p=0)**
  - Sets the current callback function for the widget.

- **unsigned int changed () const**
  - Checks if the widget value changed since the last callback.

- **void clear_active ()**
  - Marks the widget as inactive without sending events or changing focus.

- **void clear_changed ()**
  - Marks the value of the widget as unchanged.

- **void clear_damage (uchar c=0)**
  - Clears or sets the damage flags.

- **void clear_output ()**
  - Sets a widget to accept input.

- **void clear_visible ()**
  - Hides the widget.

- **void clear_visible_focus ()**
  - Disables keyboard focus navigation with this widget.

- **Fl_Color color () const**
  - Gets the background color of the widget.

- **void color (Fl_Color bg)**
  - Sets the background color of the widget.

- **void color (Fl_Color bg, Fl_Color sel)**
  - Sets the background and selection color of the widget.

- **Fl_Color color2 () const**
  - For back compatibility only.

- **void color2 (unsigned a)**
  - For back compatibility only.

- **int contains (const Fl_Widget ∗w) const**
  - Checks if w is a child of this widget.

- **void copy_label (const char ∗new_label)**
  - Sets the current label.

- **void copy_tooltip (const char ∗text)**
  - Sets the current tooltip text.

- **uchar damage () const**
  - Returns non-zero if draw() needs to be called.

- **void damage (uchar c)**
  - Sets the damage bits for the widget.

- **void damage (uchar c, int x, int y, int w, int h)**
  - Sets the damage bits for an area inside the widget.

- **int damage_resize (int, int, int, int)**
  - Internal use only.

- **void deactivate ()**
  - Deactivates the widget.
• Fl_Image * deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

• const Fl_Image * deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• int h () const
  Gets the widget height.

• virtual void hide ()
  Makes a widget invisible.

• Fl_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.

• const Fl_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.

• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.

• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.

• int image_bound () const
  Returns whether the image is managed by the widget.

• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.

• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().

• const char * label () const
  Gets the current label text.

• void label (const char *text)
  Sets the current label pointer.

• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.

• Fl_Color labelcolor () const
  Gets the label color.

• void labelcolor (Fl_Color c)
  Sets the label color.

• Fl_Font labelfont () const
  Gets the font to use.

• void labelfont (Fl_Font f)
  Sets the font to use.

• Fl_Fontsize labelsize () const
  Gets the font size to use.
Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int `takesevents`() const
  Returns if the widget is able to take events.
• int `test_shortcut`()  
  Returns true if the widget’s label contains the entered ‘&x’ shortcut.
• const char * `tooltip`() const
  Gets the current tooltip text.
• void `tooltip`(const char *text)  
  Sets the current tooltip text.
• Fl_Window * `top_window`() const
  Returns a pointer to the top-level window for the widget.
• Fl_Window * `top_window_offset`(int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar `type`() const
  Gets the widget type.
• void `type`(uchar t)  
  Sets the widget type.
• int `use_accents_menu`()  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * `user_data`() const
  Gets the user data for this widget.
• void `user_data`(Fl_Callback_User_Data *v, bool auto_free)  
  Sets the user data for this widget.
• void `user_data`(void *v)  
  Sets the user data for this widget.
• unsigned int `visible`() const
  Returns whether a widget is visible.
• unsigned int `visible_focus`() const
  Checks whether this widget has a visible focus.
• void `visible_focus`(int v)  
  Modifies keyboard focus navigation.
• int `visible_r`() const
  Returns whether a widget and all its parents are visible.
• int `w`() const
  Gets the widget width.
• Fl_When `when`() const
  Returns the conditions under which the callback is called.
• void `when`(uchar i)  
  Sets the flags used to decide when a callback is called.
• Fl_Window * `window`() const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int `x`() const
  Gets the widget position in its window.
• int `y`() const
  Gets the widget position in its window.
• virtual ~Fl_Widget()  
  Destroys the widget.

Protected Member Functions

• void `draw`() FL_OVERRIDE
  Draws the widget.
Protected Member Functions inherited from Fl_Menu_

- int item_pathname_ (char ∗name, int namelen, const Fl_Menu_Item ∗finditem, const Fl_Menu_Item ∗menu=0) const

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  
  Clears a flag in the flags mask.
- void draw_backdrop () const
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y , int W, int H) const
  
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.
- void draw_label () const
  
  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.
- void set_flag (unsigned int c)
  
  Sets a flag in the flags mask.
- void w (int v)
  
  Internal use only.
- void x (int v)
  
  Internal use only.
- void y (int v)
  
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)
  
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char ∗t)
  
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char ∗t, const bool require_alt=false)
  
  Returns true if the given text ∗t contains the entered '&x' shortcut.
12.23.1 Detailed Description

A button that is used to pop up a menu.

This is a button that, when pushed, pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects. Motif calls this an OptionButton.

The only difference between this and a Fl_Menu_Button is that the name of the most recent chosen menu item is displayed inside the box, while the label is displayed outside the box. However, since the use of this is most often to control a single variable rather than do individual callbacks, some of the Fl_Menu_Button methods are redescribed here in those terms.

When the user clicks a menu item, value() is set to that item and then:

- The item’s callback is done if one has been set; the Fl_Choice is passed as the Fl_Widget* argument, along with any userdata configured for the callback.

- If the item does not have a callback, the Fl_Choice widget’s callback is done instead, along with any userdata configured for it. The callback can determine which item was picked using value(), mvalue(), item_pathname(), etc.
All three mouse buttons pop up the menu. The Forms behavior of the first two buttons to increment/decrement the choice is not implemented. This could be added with a subclass, however.

The menu will also pop up in response to shortcuts indicated by putting a ‘&’ character in the label(). See Fl_Button::shortcut(int s) for a description of this.

Typing the shortcut() of any of the items will do exactly the same as when you pick the item with the mouse. The ‘&’ character in item names are only looked at when the menu is popped up, however.

The inherited Fl_Widget::changed() and related methods can be used as follows:

- int Fl_Widget::changed() const This value is true when the user picks a different value. It is turned off by value() and just before doing a callback (the callback can turn it back on if desired).
- void Fl_Widget::set_changed() This method sets the changed() flag.
- void Fl_Widget::clear_changed() This method clears the changed() flag.

The inherited Fl_Menu_::down_box() methods can be used as follows:

- Fl_Boxtype Fl_Menu_::down_box() const Gets the current down box, which is used when the menu is popped up. The default down box type is FL_DOWN_BOX.
- void Fl_Menu_::down_box(Fl_Boxtype b) Sets the current down box type to b.

Simple example:

```c
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Choice.H>

int main() {
  Fl_Window *win = new Fl_Window(300,200);
  Fl_Choice *choice = new Fl_Choice(100,10,100,25,"Choice:"
  choice->add("Zero");
  choice->add("One");
  choice->add("Two");
  choice->add("Three");
  choice->value(2); // make "Two" selected by default (zero based)
  win->end();
  win->show();
  return Fl::run();
}
```

### 12.23.2 Constructor & Destructor Documentation

#### 12.23.2.1 Fl_Choice()

Fl_Choice::Fl_Choice (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char * L = 0 )

Create a new Fl_Choice widget using the given position, size and label string. The default boxtype is FL_UP_BOX.

The constructor sets menu() to NULL. See Fl_Menu_ for the methods to set or change the menu.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>X,Y,W,H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

### 12.23.3 Member Function Documentation

#### 12.23.3.1 draw()

void Fl_Choice::draw ( ) [protected], [virtual]
Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as
soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded
scrollbar, you can do it (because draw() is virtual) like this:
```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```
Implements Fl_Widget.

12.23.3.2 handle()

```cpp
int Fl_Choice::handle (  
    int event ) [virtual]
```
Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-
circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class.
This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The
effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses)
in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In
such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by
returning 1 even if you didn't do anything with it.

Parameters

```
in  event  the kind of event received
```

Return values

```
0 if the event was not used or understood
1 if the event was used and can be deleted
```

See also

Fl_Event

Reimplemented from Fl_Widget.
Reimplemented in Fl_Scheme_Choice.

12.23.3.3 value() [1/3]

```cpp
int Fl_Choice::value ( ) const [inline]
```
Gets the index of the last item chosen by the user.
The index is -1 initially.

12.23.3.4 value() [2/3]

```cpp
int Fl_Choice::value (  
    const Fl_Menu_Item * v )
```
Sets the currently selected value using a pointer to menu item.
Changing the selected value causes a redraw().

Parameters

```
in  v   pointer to menu item in the menu item array.
```
Returns
non-zero if the new value is different to the old one.

12.23.3.5 value() [3/3]
int Fl_Choice::value (int v)
Sets the currently selected value using the index into the menu item array.
Changing the selected value causes a redraw().

Parameters

\[ \text{in } v \text{ index of value in the menu item array.} \]

Returns
non-zero if the new value is different to the old one.

The documentation for this class was generated from the following files:

• Fl_Choice.H
• Fl_Choice.cxx

12.24 Fl_Clock Class Reference

This widget provides a round analog clock display.

#include <Fl_Clock.H>

Inheritance diagram for Fl_Clock:

```plaintext
Fl_Widget
  Fl_Clock_Output
  Fl_Clock
  Fl_Round_Clock
```

Public Member Functions

- **Fl_Clock** (int X, int Y, int W, int H, const char *L=0)
  
  Create an Fl_Clock widget using the given position, size, and label string.

- **Fl_Clock** (uchar t, int X, int Y, int W, int H, const char *L)
  
  Create an Fl_Clock widget using the given clock type \( t \), position, size, and label string.

- int handle (int) FL_OVERRIDE
  
  Handles the specified event.

- ~Fl_Clock ()
  
  The destructor removes the clock.

Public Member Functions inherited from Fl_Clock_Output

- **Fl_Clock_Output** (int X, int Y, int W, int H, const char *L=0)
  
  Create a new Fl_Clock_Output widget with the given position, size and label.
• int \texttt{hour} () const
  \textit{Returns the displayed hour (0 to 23).}
• int \texttt{minute} () const
  \textit{Returns the displayed minute (0 to 59).}
• int \texttt{second} () const
  \textit{Returns the displayed second (0 to 60, 60=leap second).}
• int \texttt{shadow} () const
  \textit{Returns the shadow drawing mode of the hands.}
• void \texttt{shadow} (int mode)
  \textit{Sets the shadow drawing mode of the hands.}
• ulong \texttt{value} () const
  \textit{Returns the displayed time.}
• void \texttt{value} (int H, int m, int s)
  \textit{Set the displayed time.}
• void \texttt{value} (ulong v)
  \textit{Set the displayed time.}

\textbf{Public Member Functions inherited from Fl_Widget}

• void \texttt{_clear_fullscreen} ()
• void \texttt{_set_fullscreen} ()
• void \texttt{activate} ()
  \textit{Activates the widget.}
• unsigned int \texttt{active} () const
  \textit{Returns whether the widget is active.}
• int \texttt{active}_r () const
  \textit{Returns whether the widget and all of its parents are active.}
• Fl\_Align \texttt{align} () const
  \textit{Gets the label alignment.}
• void \texttt{align} (Fl\_Align alignment)
  \textit{Sets the label alignment.}
• long \texttt{argument} () const
  \textit{Gets the current user data (long) argument that is passed to the callback function.}
• void \texttt{argument} (long v)
  \textit{Sets the current user data (long) argument that is passed to the callback function.}
• virtual class Fl\_Gl\_Window \ast \texttt{as_gl_window} ()
  \textit{Returns an Fl\_Gl\_Window pointer if this widget is an Fl\_Gl\_Window.}
• virtual class Fl\_Gl\_Window const \ast \texttt{as_gl_window} () const
• virtual Fl\_Group \ast \texttt{as_group} ()
  \textit{Returns an Fl\_Group pointer if this widget is an Fl\_Group.}
• virtual Fl\_Group const \ast \texttt{as_group} () const
• virtual Fl\_Window \ast \texttt{as_window} ()
  \textit{Returns an Fl\_Window pointer if this widget is an Fl\_Window.}
• virtual Fl\_Window const \ast \texttt{as_window} () const
• void \texttt{bind_deimage} (Fl\_Image \ast img)
  \textit{Sets the image to use as part of the widget label when in the inactive state.}
• void \texttt{bind_deimage} (int f)
  \textit{Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.}
• void \texttt{bind_image} (Fl\_Image \ast img)
  \textit{Sets the image to use as part of the widget label when in the active state.}
• void \texttt{bind_image} (int f)
Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype box () const**
  Gets the box type of the widget.
- **void box (Fl_Boxtype new_box)**
  Sets the box type for the widget.
- **Fl_Callback_p callback () const**
  Gets the current callback function for the widget.
- **void callback (Fl_Callback *cb)**
  Sets the current callback function for the widget.
- **void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)**
  Sets the current callback function and managed user data for the widget.
- **void callback (Fl_Callback *cb, void *p)**
  Sets the current callback function and data for the widget.
- **void callback (Fl_Callback0 *cb)**
  Sets the current callback function for the widget.
- **void callback (Fl_Callback1 *cb, long p=0)**
  Sets the current callback function for the widget.
- **unsigned int changed () const**
  Checks if the widget value changed since the last callback.
- **void clear_active ()**
  Marks the widget as inactive without sending events or changing focus.
- **void clear_changed ()**
  Marks the value of the widget as unchanged.
- **void clear_damage (uchar c=0)**
  Clears or sets the damage flags.
- **void clear_output ()**
  Sets a widget to accept input.
- **void clear_visible ()**
  Hides the widget.
- **void clear_visible_focus ()**
  Disables keyboard focus navigation with this widget.
- **Fl_Color color () const**
  Gets the background color of the widget.
- **void color (Fl_Color bg)**
  Sets the background color of the widget.
- **void color (Fl_Color bg, Fl_Color sel)**
  Sets the background and selection color of the widget.
- **Fl_Color color2 () const**
  For back compatibility only.
- **void color2 (unsigned a)**
  For back compatibility only.
- **int contains (const Fl_Widget *w) const**
  Checks if w is a child of this widget.
- **void copy_label (const char *new_label)**
  Sets the current label.
- **void copy_tooltip (const char *text)**
  Sets the current tooltip text.
- **uchar damage () const**
  Returns non-zero if draw() needs to be called.
- **void damage (uchar c)**
  Sets the damage bits for the widget.
• void **damage**(uchar c, int x, int y, int w, int h)
  
  \textit{Sets the damage bits for an area inside the widget.}

• void **deactivate** ()
  
  \textit{Deactivates the widget.}

• Fl\_Image \* **deimage** ()
  
  \textit{Gets the image that is used as part of the widget label when in the inactive state.}

• const Fl\_Image \* **deimage** () const
  
  \textit{Gets the image that is used as part of the widget label when in the inactive state.}

• void **deimage** (Fl\_Image \&img)
  
  \textit{Sets the image to use as part of the widget label when in the inactive state.}

• void **deimage** (Fl\_Image \*img)
  
  \textit{Sets the image to use as part of the widget label when in the inactive state.}

• int **deimage\_bound** () const
  
  \textit{Returns whether the inactive image is managed by the widget.}

• void **do\_callback**(Fl\_Callback\_Reason reason=FL\_REASON\_UNKNOWN)
  
  \textit{Calls the widget callback function with default arguments.}

• void **do\_callback**(Fl\_Widget \*widget, long arg, Fl\_Callback\_Reason reason=FL\_REASON\_UNKNOWN)
  
  \textit{Calls the widget callback function with arbitrary arguments.}

• void **do\_callback**(Fl\_Widget \*widget, void \*arg=0, Fl\_Callback\_Reason reason=FL\_REASON\_UNKNOWN)
  
  \textit{Calls the widget callback function with arbitrary arguments.}

• void **draw\_label**(int, int, int, int, Fl\_Align) const
  
  \textit{Draws the label in an arbitrary bounding box with an arbitrary alignment.}

• int **h** () const
  
  \textit{Gets the widget height.}

• virtual void **hide** ()
  
  \textit{Makes a widget invisible.}

• Fl\_Image \* **image** ()
  
  \textit{Gets the image that is used as part of the widget label when in the active state.}

• const Fl\_Image \* **image** () const
  
  \textit{Gets the image that is used as part of the widget label when in the active state.}

• void **image** (Fl\_Image \&img)
  
  \textit{Sets the image to use as part of the widget label when in the active state.}

• void **image** (Fl\_Image \*img)
  
  \textit{Sets the image to use as part of the widget label when in the active state.}

• int **image\_bound** () const
  
  \textit{Returns whether the image is managed by the widget.}

• int **inside**(const Fl\_Widget \*wgt) const
  
  \textit{Checks if this widget is a child of wgt.}

• int **is\_label\_copied** () const
  
  \textit{Returns whether the current label was assigned with copy\_label().}

• const char \* **label** () const
  
  \textit{Gets the current label text.}

• void **label** (const char \*text)
  
  \textit{Sets the current label pointer.}

• void **label** (Fl\_Labeltype a, const char \*b)
  
  \textit{Shortcut to set the label text and type in one call.}

• Fl\_Color **labelcolor** () const
  
  \textit{Gets the label color.}

• void **labelcolor** (Fl\_Color c)
Sets the label color.

- `Fl_Font labelfont () const`
  Gets the font to use.

- `void labelfont (Fl_Font f)`
  Sets the font to use.

- `Fl_Fontsize labelsize () const`
  Gets the font size in pixels.

- `void labelsize (Fl_Fontsize pix)`
  Sets the font size in pixels.

- `Fl_Labeltype labeltype () const`
  Gets the label type.

- `void labeltype (Fl_Labeltype a)`
  Sets the label type.

- `void measure_label (int &ww, int &hh) const`
  Sets width ww and height hh accordingly with the label size.

- `bool needs_keyboard () const`
  Returns whether this widget needs a keyboard.

- `void needs_keyboard (bool needs)`
  Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  Returns if a widget is used for output only.

- `Fl_Group * parent () const`
  Returns a pointer to the parent widget.

- `void parent (Fl_Group *p)`
  Internal use only - “for hacks only”.

- `void position (int X, int Y)`
  Repositions the window or widget.

- `void redraw ()`
  Schedules the drawing of the widget.

- `void redraw_label ()`
  Schedules the drawing of the label.

- `virtual void resize (int x, int y, int w, int h)`
  Changes the size or position of the widget.

- `Fl_Color selection_color () const`
  Gets the selection color.

- `void selection_color (Fl_Color a)`
  Sets the selection color.

- `void set_active ()`
  Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  Marks the value of the widget as changed.

- `void set_output ()`
  Sets a widget to output only.

- `void set_visible ()`
  Makes the widget visible.

- `void set_visible_focus ()`
  Enables keyboard focus navigation with this widget.

- `int shortcut_label () const`
  Returns whether the widget's label uses ‘&’ to indicate shortcuts.

- `void shortcut_label (int value)`
  Sets whether the widget's label uses ‘&’ to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered 'x' shortcut.
• const char * tooltip () const
  Gets the current tooltip text.
• void tooltip (const char *text)
  Sets the current tooltip text.
• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * user_data () const
  Gets the user data for this widget.
• void user_data (FL_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• FL_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.
Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Clock_Output

- void draw () FL_OVERRIDE
  Draw clock with current position and size.
- void draw (int X, int Y, int W, int H)
  Draw clock with the given position and size.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void draw_label () const
  Draws the widget's label at the defined label position.
12.24 Fl_Clock Class Reference

- void **draw_label** (int, int, int, int) const  
  
  *Draws the label in an arbitrary bounding box.*

- **Fl_Widget** (int x, int y, int w, int h, const char ∗label=0L)  
  
  *Creates a widget at the given position and size.*

- unsigned int **flags** () const  
  
  *Gets the widget flags mask.*

- void **h** (int v)  
  
  *Internal use only.*

- void **set_flag** (unsigned int c)  
  
  *Sets a flag in the flags mask.*

- void **w** (int v)  
  
  *Internal use only.*

- void **x** (int v)  
  
  *Internal use only.*

- void **y** (int v)  
  
  *Internal use only.*

12.24.1 Detailed Description

This widget provides a round analog clock display. **Fl_Clock** is provided for Forms compatibility. It installs a 1-second timeout callback using Fl::add_timeout(). You can choose the rounded or square type of the clock with **type()**. Please see **Fl_Clock_Output** widget for applicable values.

---

See also

- class **Fl_Clock_Output**
## 12.24.2 Constructor & Destructor Documentation

### 12.24.2.1 Fl_Clock() [1/2]

```cpp
Fl_Clock::Fl_Clock (int X, int Y, int W, int H, const char ∗L = 0)
```

Create an Fl_Clock widget using the given position, size, and label string.
The default clock type is FL_SQUARE_CLOCK and the default boxtype is FL_UP_BOX.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X, Y, W, H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

### 12.24.2.2 Fl_Clock() [2/2]

```cpp
Fl_Clock::Fl_Clock (uchar t, int X, int Y, int W, int H, const char ∗L)
```

Create an Fl_Clock widget using the given clock type t, position, size, and label string.
The default clock type t is FL_SQUARE_CLOCK. You can set the clock type to FL_ROUND_CLOCK or any other valid clock type. See Fl_Clock_Output widget for applicable values.
The default boxtype is FL_UP_BOX for FL_SQUARE_CLOCK and FL_NO_BOX for FL_ROUND_CLOCK, if set by the constructor. If you change the clock type with type() later you should also set the boxtype with box().

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>type of clock: FL_ROUND_CLOCK or FL_SQUARE_CLOCK (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X, Y, W, H</td>
<td>position and size of the widget</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

See also
class Fl_Clock_Output

## 12.24.3 Member Function Documentation

### 12.24.3.1 handle()

```cpp
int Fl_Clock::handle (int event) [virtual]
```

Handles the specified event.
You normally don’t call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don’t short-circuit events that you don’t handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don’t really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses)
in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>event</th>
<th>the kind of event received</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>if the event was not used or understood</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>

See also

Fl_Event

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

- Fl_Clock.H
- Fl_Clock.cxx

12.25 Fl_Clock_Output Class Reference

This widget can be used to display a program-supplied time.

#include <Fl_Clock.H>

Inheritance diagram for Fl_Clock_Output:

```
Fl_Widget
  ↓
Fl_Clock_Output
  ↓
Fl_Clock
  ↓
Fl_Round_Clock
```

Public Member Functions

- Fl_Clock_Output (int X, int Y, int W, int H, const char *L=0)
  Create a new Fl_Clock_Output widget with the given position, size and label.
- int hour () const
  Returns the displayed hour (0 to 23).
- int minute () const
  Returns the displayed minute (0 to 59).
- int second () const
  Returns the displayed second (0 to 60, 60=leap second).
- int shadow () const
  Returns the shadow drawing mode of the hands.
- void shadow (int mode)
  Sets the shadow drawing mode of the hands.
• **ulong value () const**
  Returns the displayed time.

• **void value (int H, int m, int s)**
  Set the displayed time.

• **void value (ulong v)**
  Set the displayed time.

### Public Member Functions inherited from Fl_Widget

- **void _clear_fullscreen ()**
  Activates the widget.

- **void _set_fullscreen ()**

- **void activate ()**
  Activates the widget.

- **unsigned int active () const**
  Returns whether the widget is active.

- **int active_r () const**
  Returns whether the widget and all of its parents are active.

- **Fl_Align align () const**
  Gets the label alignment.

- **void align (Fl_Align alignment)**
  Sets the label alignment.

- **long argument () const**
  Gets the current user data (long) argument that is passed to the callback function.

- **void argument (long v)**
  Sets the current user data (long) argument that is passed to the callback function.

- **virtual class Fl_Gl_Window ∗ as_gl_window ()**
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- **virtual class Fl_Gl_Window const ∗ as_gl_window () const**

- **virtual Fl_Group ∗ as_group ()**
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- **virtual Fl_Group const ∗ as_group () const**

- **virtual Fl_Window ∗ as_window ()**
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- **virtual Fl_Window const ∗ as_window () const**

- **void bind_deimage (Fl_Image ∗ img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **void bind_deimage (int f)**
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- **void bind_image (Fl_Image ∗ img)**
  Sets the image to use as part of the widget label when in the active state.

- **void bind_image (int f)**
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype box () const**
  Gets the box type of the widget.

- **void box (Fl_Boxtype new_box)**
  Sets the box type for the widget.

- **Fl_Callback_p callback () const**
  Gets the current callback function for the widget.

- **void callback (Fl_Callback ∗cb)**
  Sets the current callback function for the widget.

- **void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)**
Sets the current callback function and managed user data for the widget.

- void `callback (Fl_Callback ∗ cb, void ∗ p)`
  Sets the current callback function and data for the widget.
- void `callback (Fl_Callback0 ∗ cb)`
  Sets the current callback function for the widget.
- void `callback (Fl_Callback1 ∗ cb, long p=0)`
  Sets the current callback function for the widget.
- unsigned int `changed () const`
  Checks if the widget value changed since the last callback.
- void `clear_active ()`
  Marks the widget as inactive without sending events or changing focus.
- void `clear_changed ()`
  Marks the value of the widget as unchanged.
- void `clear_damage (uchar c=0)`
  Clears or sets the damage flags.
- void `clear_output ()`
  Sets a widget to accept input.
- void `clear_visible ()`
  Hides the widget.
- void `clear_visible_focus ()`
  Disables keyboard focus navigation with this widget.
- Fl_Color `color () const`
  Gets the background color of the widget.
- void `color (Fl_Color bg)`
  Sets the background color of the widget.
- void `color (Fl_Color bg, Fl_Color sel)`
  Sets the background and selection color of the widget.
- Fl_Color `color2 () const`
  For back compatibility only.
- void `color2 (unsigned a)`
  For back compatibility only.
- int `contains (const Fl_Widget ∗ w) const`
  Checks if w is a child of this widget.
- void `copy_label (const char ∗ new_label)`
  Sets the current label.
- void `copy_tooltip (const char ∗ text)`
  Sets the current tooltip text.
- uchar `damage () const`
  Returns non-zero if `draw()` needs to be called.
- void `damage (uchar c)`
  Sets the damage bits for the widget.
- void `damage (uchar c, int x, int y, int w, int h)`
  Sets the damage bits for an area inside the widget.
- int `damage_resize (int, int, int, int)`
  Internal use only.
- void `deactivate ()`
  Deactivates the widget.
- Fl_Image ∗ `deimage ()`
  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image ∗ `deimage () const`
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual int handle (int event)
  Handles the specified event.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image *image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image *image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
Sets the font size in pixels.

- `Fl_Labeltype labeltype () const`
  Gets the label type.

- `void labeltype (Fl_Labeltype a)`
  Sets the label type.

- `void measure_label (int &ww, int &hh) const`
  Sets width `ww` and height `hh` accordingly with the label size.

- `bool needs_keyboard () const`
  Returns whether this widget needs a keyboard.

- `void needs_keyboard (bool needs)`
  Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  Returns if a widget is used for output only.

- `Fl_Group * parent () const`
  Returns a pointer to the parent widget.

- `void parent (Fl_Group * p)`
  Internal use only - “for hacks only”.

- `void position (int X, int Y)`
  Repositions the window or widget.

- `void redraw ()`
  Schedules the drawing of the widget.

- `void redraw_label ()`
  Schedules the drawing of the label.

- `virtual void resize (int x, int y, int w, int h)`
  Changes the size or position of the widget.

- `Fl_Color selection_color () const`
  Gets the selection color.

- `void selection_color (Fl_Color a)`
  Sets the selection color.

- `void set_active ()`
  Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  Marks the value of the widget as changed.

- `void set_output ()`
  Sets a widget to output only.

- `void set_visible ()`
  Makes the widget visible.

- `void set_visible_focus ()`
  Enables keyboard focus navigation with this widget.

- `int shortcut_label () const`
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.

- `void shortcut_label (int value)`
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.

- `virtual void show ()`
  Makes a widget visible.

- `void size (int W, int H)`
  Changes the size of the widget.

- `int take_focus ()`
  Gives the widget the keyboard focus.

- `unsigned int takesevents () const`
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.

• const char * tooltip () const
  Gets the current tooltip text.

• void tooltip (const char *text)
  Sets the current tooltip text.

• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const
  Gets the widget type.

• void type (uchar t)
  Sets the widget type.

• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void * user_data () const
  Gets the user data for this widget.

• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

• void user_data (void *v)
  Sets the user data for this widget.

• unsigned int visible () const
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

• void visible_focus (int v)
  Modifies keyboard focus navigation.

• int visible_r () const
  Returns whether a widget and all its parents are visible.

• int w () const
  Gets the widget width.

• Fl_When when () const
  Returns the conditions under which the callback is called.

• void when (uchar i)
  Sets the flags used to decide when a callback is called.

• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  Gets the widget position in its window.

• int y () const
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  Destroys the widget.

Protected Member Functions

• void draw () FL_OVERRIDE
  Draw clock with current position and size.

• void draw (int X, int Y, int W, int H)
  Draw clock with the given position and size.
Protected Member Functions inherited from Fl_Widget

- void **clear_flag**(unsigned int c)
  Clears a flag in the flags mask.

- void **draw_backdrop**() const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void **draw_box**() const
  Draws the widget box according its box style.

- void **draw_box**(Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

- void **draw_box**(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- void **draw_focus**() const
  Draws a focus rectangle around the widget.

- void **draw_focus**(Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

- void **draw_focus**(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

- void **draw_label**() const
  Draws the widget's label at the defined label position.

- void **draw_label**(int, int, int, int) const
  Draws the label in an arbitrary bounding box.

- Fl_Widget(int x, int y, int w, int h, const char ∗ label=0L)
  Creates a widget at the given position and size.

- unsigned int **flags**() const
  Gets the widget flags mask.

- void **h**(int v)
  Internal use only.

- void **set_flag**(unsigned int c)
  Sets a flag in the flags mask.

- void **w**(int v)
  Internal use only.

- void **x**(int v)
  Internal use only.

- void **y**(int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void **default_callback**(Fl_Widget ∗ widget, void ∗ data)
  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut(const char ∗ t)
  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut(const char ∗ t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.
Protected Types inherited from **Fl_Widget**

- `enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<25, USERFLAG2 = 1<<26, USERFLAG1 = 1<<27
} flags possible values enumeration.

### 12.25.1 Detailed Description

This widget can be used to display a program-supplied time. The time shown on the clock is not updated. To display the current time, use `Fl_Clock` instead.

![Figure 12.11 FL_SQUARE_CLOCK type](image)

**Figure 12.11 FL_SQUARE_CLOCK type**

![Figure 12.12 FL_ROUND_CLOCK type](image)

**Figure 12.12 FL_ROUND_CLOCK type**

Values for clock `type()` (#include `<FL/Clock.H>`):

```c
#define FL_SQUARE_CLOCK 0 // Square Clock variant
#define FL_ROUND_CLOCK 1  // Round Clock variant
#define FL_ANALOG_CLOCK FL_SQUARE_CLOCK // An analog clock is square
#define FL_DIGITAL_CLOCK FL_SQUARE_CLOCK // Not yet implemented
```

### 12.25.2 Constructor & Destructor Documentation

#### 12.25.2.1 Fl_Clock_Output()

```c
Fl_Clock_Output::Fl_Clock_Output ( 
```
Create a new `Fl_Clock_Output` widget with the given position, size and label. The default clock type is `FL_SQUARE_CLOCK` and the default boxtype is `FL_UP_BOX`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>X, Y, W, H</code></td>
<td>position and size of the widget</td>
</tr>
<tr>
<td><code>L</code></td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

### 12.25.3 Member Function Documentation

#### 12.25.3.1 draw() [1/2]

```cpp
void Fl_Clock_Output::draw () [protected], [virtual]
```

Draw clock with current position and size. Implements `Fl_Widget`.

#### 12.25.3.2 draw() [2/2]

```cpp
void Fl_Clock_Output::draw ( int X, int Y, int W, int H ) [protected]
```

Draw clock with the given position and size.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>X, Y, W, H</code></td>
<td>position and size</td>
</tr>
</tbody>
</table>

#### 12.25.3.3 hour()

```cpp
int Fl_Clock_Output::hour ( ) const [inline]
```

Returns the displayed hour (0 to 23).

See also

- `value()`, `minute()`, `second()`

#### 12.25.3.4 minute()

```cpp
int Fl_Clock_Output::minute ( ) const [inline]
```

Returns the displayed minute (0 to 59).

See also

- `value()`, `hour()`, `second()`

#### 12.25.3.5 second()

```cpp
int Fl_Clock_Output::second ( ) const [inline]
```

Returns the displayed second (0 to 60, 60=leap second).
See also

\texttt{value()}, \texttt{hour()}, \texttt{minute()}

\subsection{shadow() [1/2]}

\begin{verbatim}
int Fl_Clock_Output::shadow ( ) const [inline]
\end{verbatim}

Returns the shadow drawing mode of the hands.

Returns

shadow drawing mode of the hands

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>no shadows</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>draw shadows of hands (default)</td>
</tr>
</tbody>
</table>

\subsection{shadow() [2/2]}

\begin{verbatim}
void Fl_Clock_Output::shadow ( int mode ) [inline]
\end{verbatim}

Sets the shadow drawing mode of the hands.

Enables (1) or disables (0) drawing the hands with shadows.

Values except 0 and 1 are reserved for future extensions and yield undefined behavior.

The default is to draw the shadows (1).

Parameters

\begin{verbatim}
in mode 1 = shadows (default), 0 = no shadows
\end{verbatim}

\subsection{value() [1/3]}

\begin{verbatim}
ulong Fl_Clock_Output::value ( ) const [inline]
\end{verbatim}

Returns the displayed time.

Returns the time in seconds since the UNIX epoch (January 1, 1970).

See also

\texttt{value(ulong)}

\subsection{value() [2/3]}

\begin{verbatim}
void Fl_Clock_Output::value ( int H, int m, int s )
\end{verbatim}

Set the displayed time.

Set the time in hours, minutes, and seconds.

Parameters

\begin{verbatim}
in H,m,s displayed time
\end{verbatim}
12.26 Fl_Color_Chooser Class Reference

See also

   hour(), minute(), second()

12.25.3.10 value() [3/3]

void Fl_Clock_Output::value (ulong v)

Set the displayed time.
Set the time in seconds since the UNIX epoch (January 1, 1970).

Parameters

| in  | v | seconds since epoch |

See also

   value()

The documentation for this class was generated from the following files:

- Fl_Clock.H
- Fl_Clock.cxx

12.26 Fl_Color_Chooser Class Reference

The Fl_Color_Chooser widget provides a standard RGB color chooser.

```c
#include <Fl_Color_Chooser.H>
```

Inheritance diagram for Fl_Color_Chooser:

```
Fl_Widget
    |           |
    |           |
Fl_Group
    |           |
    |           |
Fl_Color_Chooser
```

Public Member Functions

- double b () const
  
  Returns the current blue value.

- Fl_Color_Chooser (int X, int Y, int W, int H, const char *L=0)
  
  Creates a new Fl_Color_Chooser widget using the given position, size, and label string.

- double g () const
  
  Returns the current green value.

- int handle (int e) FL_OVERRIDE
  
  Handles all events received by this widget.

- int hsv (double H, double S, double V)
  
  Set the hsv values.

- double hue () const
  
  Returns the current hue.

- int mode ()
  
  Returns which Fl_Color_Chooser variant is currently active.

- void mode (int newMode)
Set which Fl_Color_Chooser variant is currently active.

- **double r () const**
  Returns the current red value.
- **int rgb (double R, double G, double B)**
  Sets the current rgb color values.
- **double saturation () const**
  Returns the saturation.
- **double value () const**
  Returns the value/brightness.

Public Member Functions inherited from Fl_Group

- **FL_Widget *&_ddfdesign_kludge ()**
  This is for forms compatibility only.
- **void add (FL_Widget &)**
  The widget is removed from its current group (if any) and then added to the end of this group.
- **void add (FL_Widget *o)**
  See void Fl_Group::add(FL_Widget &w)
- **void addResizable (FL_Widget &o)**
  Adds a widget to the group and makes it the resizable widget.
- **FL_Widget *const * array () const**
  Returns a pointer to the array of children.
- **FL_Group const * as_group () const FL_OVERRIDE**
  FL_Group * as_group () FL_OVERRIDE
  Returns an FL_Group pointer if this widget is an FL_Group.
- **void begin ()**
  Sets the current group so you can build the widget tree by just constructing the widgets.
- **FL_Widget * child (int n) const**
  Returns array()[n].
- **int children () const**
  Returns how many child widgets the group has.
- **void clear ()**
  Deletes all child widgets from memory recursively.
- **unsigned int clip_children ()**
  Returns the current clipping mode.
- **void clip_children (int c)**
  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- **virtual int delete_child (int n)**
  Removes the widget at index from the group and deletes it.
- **void end ()**
  Exactly the same as current(this->parent()).
- **int find (const FL_Widget &o) const**
  See int Fl_Group::find(const FL_Widget *w) const.
- **int find (const FL_Widget *) const**
  Searches the child array for the widget and returns the index.
- **FL_Group (int, int, int, int, const char *)=0**
  Creates a new FL_Group widget using the given position, size, and label string.
- **void focus (FL_Widget *)**
  void forms_end ()
  This is for forms compatibility only.
- **int handle (int) FL_OVERRIDE**
Handles the specified event.

- void init_sizes()
  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)
  This does insert(w, find(before)).
- void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)
  Removes the widget o from the group.
- void remove (int index)
  Removes the widget at index from the group but does not delete it.
- Fl_Widget * resizable () const
  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)
  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- void resize (int, int, int, int) FL_OVERRIDE
  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()
  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * as_gl_window () const
- virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * as_window () const
- void bind_deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- **void bind_image (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the active state.

- **void bind_image (int f)**
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype box () const**
  Gets the box type of the widget.

- **void box (Fl_Boxtype new_box)**
  Sets the box type for the widget.

- **Fl_Callback_p callback () const**
  Gets the current callback function for the widget.

- **void callback (Fl_Callback *cb)**
  Sets the current callback function for the widget.

- **void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)**
  Sets the current callback function and managed user data for the widget.

- **void callback (Fl_Callback *cb, void *p)**
  Sets the current callback function and data for the widget.

- **void callback (Fl_Callback0 *cb)**
  Sets the current callback function for the widget.

- **void callback (Fl_Callback1 *cb, long p=0)**
  Sets the current callback function for the widget.

- **unsigned int changed () const**
  Checks if the widget value changed since the last callback.

- **void clear_active ()**
  Marks the widget as inactive without sending events or changing focus.

- **void clear_changed ()**
  Marks the value of the widget as unchanged.

- **void clear_damage (uchar c=0)**
  Clears or sets the damage flags.

- **void clear_output ()**
  Sets a widget to accept input.

- **void clear_visible ()**
  Hides the widget.

- **void clear_visible_focus ()**
  Disables keyboard focus navigation with this widget.

- **Fl_Color color () const**
  Gets the background color of the widget.

- **void color (Fl_Color bg)**
  Sets the background color of the widget.

- **void color (Fl_Color bg, Fl_Color sel)**
  Sets the background and selection color of the widget.

- **Fl_Color color2 () const**
  For back compatibility only.

- **void color2 (unsigned a)**
  For back compatibility only.

- **int contains (const Fl_Widget *w) const**
  Checks if w is a child of this widget.

- **void copy_label (const char *new_label)**
  Sets the current label.

- **void copy_tooltip (const char *text)**
  Sets the current tooltip text.
- **uchar damage () const**
  
  Returns non-zero if `draw()` needs to be called.

- **void damage (uchar c)**

  Sets the damage bits for the widget.

- **void damage (uchar c, int x, int y, int w, int h)**

  Sets the damage bits for an area inside the widget.

- **int damage_resize (int, int, int, int)**

  Internal use only.

- **void deactivate ()**

  Deactivates the widget.

- **Fl_Image ∗ deimage ()**

  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image ∗ deimage () const**

  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (Fl_Image &img)**

  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (Fl_Image ∗ img)**

  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const**

  Returns whether the inactive image is managed by the widget.

- **void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**

  Calls the widget callback function with default arguments.

- **void do_callback (Fl_Widget ∗ widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**

  Calls the widget callback function with arbitrary arguments.

- **void do_callback (Fl_Widget ∗ widget, void ∗ arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**

  Calls the widget callback function with arbitrary arguments.

- **void draw_label (int, int, int, int, Fl_Align) const**

  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h () const**

  Gets the widget height.

- **virtual void hide ()**

  Makes a widget invisible.

- **Fl_Image ∗ image ()**

  Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image ∗ image () const**

  Gets the image that is used as part of the widget label when in the active state.

- **void image (Fl_Image &img)**

  Sets the image to use as part of the widget label when in the active state.

- **void image (Fl_Image ∗ img)**

  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const**

  Returns whether the image is managed by the widget.

- **int inside (const Fl_Widget ∗ wgt) const**

  Checks if this widget is a child of `wgt`.

- **int is_label_copied () const**

  Returns whether the current label was assigned with `copy_label()`.

- **const char ∗ label () const**

  Gets the current label text.

- **void label (const char ∗ text)**

  Sets the current label pointer.

- **void label (Fl_Labeltype a, const char ∗ b)**
Shortcut to set the label text and type in one call.

- **FL_Color labelcolor () const**
  
  Gets the label color.

- **void labelcolor (FL_Color c)**
  
  Sets the label color.

- **FL_Font labelfont () const**
  
  Gets the font to use.

- **void labelfont (FL_Font f)**
  
  Sets the font to use.

- **FL_Fontsize labelsize () const**
  
  Gets the font size in pixels.

- **void labelsize (FL_Fontsize pix)**
  
  Sets the font size in pixels.

- **FL_Labeltype labeltype () const**
  
  Gets the label type.

- **void labeltype (FL_Labeltype a)**
  
  Sets the label type.

- **void measure_label (int &ww, int &hh) const**
  
  Sets width ww and height hh accordingly with the label size.

- **bool needs_keyboard () const**
  
  Returns whether this widget needs a keyboard.

- **void needs_keyboard (bool needs)**
  
  Sets whether this widget needs a keyboard.

- **unsigned int output () const**
  
  Returns if a widget is used for output only.

- **FL_Group *parent () const**
  
  Returns a pointer to the parent widget.

- **void parent (FL_Group *p)**
  
  Internal use only - "for hacks only".

- **void position (int X, int Y)**
  
  Repositions the window or widget.

- **void redraw ()**
  
  Schedules the drawing of the widget.

- **void redraw_label ()**
  
  Schedules the drawing of the label.

- **FL_Color selection_color () const**
  
  Gets the selection color.

- **void selection_color (FL_Color a)**
  
  Sets the selection color.

- **void set_active ()**
  
  Marks the widget as active without sending events or changing focus.

- **void set_changed ()**
  
  Marks the value of the widget as changed.

- **void set_output ()**
  
  Sets a widget to output only.

- **void set_visible ()**
  
  Makes the widget visible.

- **void set_visible_focus ()**
  
  Enables keyboard focus navigation with this widget.

- **int shortcut_label () const**
  
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '& to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered 'x' shortcut.
• const char * tooltip () const
  Gets the current tooltip text.
• void tooltip (const char *text)
  Sets the current tooltip text.
• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void *user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.
Static Public Member Functions

- static void hsv2rgb (double H, double S, double V, double &R, double &G, double &B)
  This static method converts HSV colors to RGB colorspace.
- static void rgb2hsv (double R, double G, double B, double &H, double &S, double &V)
  This static method converts RGB colors to HSV colorspace.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()
  Returns the currently active group.
- static void current (Fl_Group *g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Related Symbols

(Note that these are not member symbols.)

- int fl_color_chooser (const char *name, double &r, double &g, double &b, int cmode)
  Pops up a window to let the user pick an arbitrary RGB color.
- int fl_color_chooser (const char *name, uchar &r, uchar &g, uchar &b, int cmode)
  Pops up a window to let the user pick an arbitrary RGB color.

Additional Inherited Members

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7
  , OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11
  , MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15
  , GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19 , 
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22 , 
  AUTO_DELETE_USER_DATA = 1<<23 , 
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 
} flags possible values enumeration.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
- void draw () FL_OVERRID
12.26 Fl_ColorChooser Class Reference

Draws the widget.

- void **draw_child** (Fl_Widget &widget) const
  Forces a child to redraw.

- void **draw_children**()
  Draws all children of the group.

- void **draw_outside_label** (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.

- virtual int **on_insert** (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.

- virtual int **on_move** (int, int)
  Allow derived groups to act when a widget is moved within the group.

- virtual void **on_remove** (int)
  Allow derived groups to act when a child widget is removed from the group.

- int * **sizes**()
  Returns the internal array of widget sizes and positions.

- void **update_child** (Fl_Widget &widget) const
  Draws a child only if it needs it.

**Protected Member Functions inherited from Fl_Widget**

- void **clear_flag** (unsigned int c)
  Clears a flag in the flags mask.

- void **draw_backdrop** () const
  If FLALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void **draw_box** () const
  Draws the widget box according its box style.

- void **draw_box** (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- void **draw_focus** () const
  Draws a focus rectangle around the widget.

- void **draw_focus** (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

- void **draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

- void **draw_label** () const
  Draws the widget's label at the defined label position.

- void **draw_label** (int, int, int, int) const
  Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.

- unsigned int **flags** () const
  Gets the widget flags mask.

- void **h** (int v)
  Internal use only.

- void **set_flag** (unsigned int c)
  Sets a flag in the flags mask.

- void **w** (int v)
  Internal use only.

- void **x** (int v)
  Internal use only.

- void **y** (int v)
  Internal use only.
12.26.1 Detailed Description

The **Fl_Color_Chooser** widget provides a standard RGB color chooser.

![Fl_Color_Chooser](image)

You can place any number of the widgets into a panel of your own design. The diagram shows the widget as part of a color chooser dialog created by the `fl_color_chooser()` function. The **Fl_Color_Chooser** widget contains the hue box, value slider, and rgb input fields from the above diagram (it does not have the color chips or the Cancel or OK buttons). The callback is done every time the user changes the rgb value. It is not done if they move the hue control in a way that produces the same rgb value, such as when saturation or value is zero.

The `fl_color_chooser()` function pops up a window to let the user pick an arbitrary RGB color. They can pick the hue and saturation in the "hue box" on the left (hold down CTRL to just change the saturation), and the brightness using the vertical slider. Or they can type the 8-bit numbers into the RGB **Fl_Value_Input** fields, or drag the mouse across them to adjust them. The pull-down menu lets the user set the input fields to show RGB, HSV, or 8-bit RGB (0 to 255).

The user can press CTRL-C to copy the currently selected color value as text in RGB hex format with leading zeroes to the clipboard, for instance `FL_GREEN` would be '00FF00' (since FLTK 1.4.0).

`fl_color_chooser()` returns non-zero if the user picks ok, and updates the RGB values. If the user picks cancel or closes the window this returns zero and leaves RGB unchanged.

If you use the color chooser on an 8-bit screen, it will allocate all the available colors, leaving you no space to exactly represent the color the user picks! You can however use `fl_rectf()` to fill a region with a simulated color using dithering.

Callback reasons can be **FL_REASON_DRAGGED**, **FL_REASON_CHANGED**, or **FL_REASON_RESELECTED**.

12.26.2 Constructor & Destructor Documentation

12.26.2.1 Fl_Color_Chooser()

```cpp
Fl_Color_Chooser::Fl_Color_Chooser (int X, int Y, int W, int H, const char * L = 0 )
```

Creates a new **Fl_Color_Chooser** widget using the given position, size, and label string.

The recommended dimensions are 200x95. The color is initialized to black.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>X, Y, W, H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>
### Member Function Documentation

#### b()

```cpp
double Fl_Color_Chooser::b ( ) const [inline]
```

Returns the current blue value.
0 <= b <= 1.

#### g()

```cpp
double Fl_Color_Chooser::g ( ) const [inline]
```

Returns the current green value.
0 <= g <= 1.

#### handle()

```cpp
int Fl_Color_Chooser::handle ( int e ) [virtual]
```

Handles all events received by this widget.
This specific `handle()` method processes the standard 'copy' function as seen in other input widgets. It copies the current color value to the clipboard as a string in RGB format ('RRGGBB'). This format is independent of the `Fl_Color_Chooser` display format setting. No other formats are supplied. The keyboard events handled are:

- `ctrl-c`
- `ctrl-x`
- `ctrl-Insert`

All other events are processed by the parent class `Fl_Group`. This enables the user to choose a color value, press `ctrl-c` to copy the value to the clipboard and paste it into a color selection widget in another application window or any other text input (e.g. a preferences dialog or an editor).

**Note**

Keyboard event handling by the current focus widget has priority, hence moving the focus to one of the buttons or selecting text in one of the input widgets effectively disables this special method.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>e</th>
<th>current event</th>
</tr>
</thead>
</table>

**Returns**

1 if event has been handled, 0 otherwise

**See also**

`Fl_Group::handle(int)`

Reimplemented from `Fl_Widget`.

#### hsv()

```cpp
int Fl_Color_Chooser::hsv ( double H, double S, double V )
```

Set the hsv values.
The passed values are clamped (or for hue, modulus 6 is used) to get legal values. Does not do the callback.
Parameters

**in** | **H,S,V** color components.

**Returns**

1 if a new hsv value was set, 0 if the hsv value was the previous one.

### 12.26.3.5 hsv2rgb()

```cpp
void Fl_Color_Chooser::hsv2rgb ( double H, double S, double V, double & R, double & G, double & B ) [static]
```

This *static* method converts HSV colors to RGB colorspace.

Parameters

**in** | **H,S,V** color components
| **out** | **R,G,B** color components

### 12.26.3.6 hue()

```cpp
double Fl_Color_Chooser::hue ( ) const [inline]
```

Returns the current hue.

0 <= hue < 6. Zero is red, one is yellow, two is green, etc. *This value is convenient for the internal calculations - some other systems consider hue to run from zero to one, or from 0 to 360.*

### 12.26.3.7 mode() [1/2]

```cpp
int Fl_Color_Chooser::mode ( ) [inline]
```

Returns which **Fl_Color_Chooser** variant is currently active.

Returns

color modes are rgb(0), byte(1), hex(2), or hsv(3)

### 12.26.3.8 mode() [2/2]

```cpp
void Fl_Color_Chooser::mode ( int newMode )
```

Set which **Fl_Color_Chooser** variant is currently active.

Parameters

| **in** | **newMode** color modes are rgb(0), byte(1), hex(2), or hsv(3) |

### 12.26.3.9 r()

```cpp
double Fl_Color_Chooser::r ( ) const [inline]
```

Returns the current red value.

0 <= r <= 1.
12.26.3.10 rgb()

```cpp
int Fl_Color_Chooser::rgb ( double R, double G, double B )
```

Sets the current rgb color values. Does not do the callback. Does not clamp (but out of range values will produce psychedelic effects in the hue selector).

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>R,G,B</td>
</tr>
</tbody>
</table>

**Returns**

1 if a new rgb value was set, 0 if the rgb value was the previous one.

12.26.3.11 rgb2hsv()

```cpp
void Fl_Color_Chooser::rgb2hsv ( double R, double G, double B, double & H, double & S, double & V ) [static]
```

This static method converts RGB colors to HSV colorspace.

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>R,G,B</td>
</tr>
<tr>
<td>out</td>
<td>H,S,V</td>
</tr>
</tbody>
</table>

12.26.3.12 saturation()

```cpp
double Fl_Color_Chooser::saturation ( ) const [inline]
```

Returns the saturation.

0 \leq saturation \leq 1.

12.26.3.13 value()

```cpp
double Fl_Color_Chooser::value ( ) const [inline]
```

Returns the value/brightness.

0 \leq value \leq 1.

The documentation for this class was generated from the following files:

- Fl_Color_Chooser.H
- Fl_Color_Chooser.cxx

12.27 Fl_Copy_Surface Class Reference

Supports copying of graphical data to the clipboard.

```cpp
#include <Fl_Copy_Surface.H>
```

Inheritance diagram for Fl_Copy_Surface:

---

Generated by Doxygen
Public Member Functions

- **Fl_Copy_Surface (int w, int h)**
  - the constructor
- **int h ()**
  - Returns the pixel height of the copy surface.
- **bool is_current () FL_OVERRIDE**
  - Is this surface the current drawing surface?
- **void origin (int *x, int *y) FL_OVERRIDE**
  - Computes the coordinates of the current origin of graphics functions.
- **void origin (int x, int y) FL_OVERRIDE**
  - Sets the position of the origin of graphics in the drawable part of the drawing surface.
- **int printable_rect (int *w, int *h) FL_OVERRIDE**
  - Computes the width and height of the drawable area of the drawing surface.
- **void set_current () FL_OVERRIDE**
  - Make this surface the current drawing surface.
- **int w ()**
  - Returns the pixel width of the copy surface.
- **~Fl_Copy_Surface ()**
  - the destructor

Public Member Functions inherited from **Fl_Widget_Surface**

- **void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)**
  - Draws the widget on the drawing surface.
- **void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)**
  - Draws a window with its title bar and frame if any.
- **void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)**
  - Draws a rectangular part of an on-screen window.

Public Member Functions inherited from **Fl_Surface_Device**

- **Fl_Graphics_Driver * driver ()**
  - Returns the graphics driver of this drawing surface.
- **virtual ~Fl_Surface_Device ()**
  - The destructor.

Protected Member Functions

- **void translate (int x, int y) FL_OVERRIDE**
  - Translates the current graphics origin accounting for the current rotation.
- **void untranslate () FL_OVERRIDE**
  - Undoes the effect of a previous translate() call.
12.27 Fl_Copy_Surface Class Reference

Protected Member Functions inherited from Fl_Widget_Surface

- Fl_Widget_Surface (Fl_Graphics_Driver *d)
  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- void driver (Fl_Graphics_Driver *graphics_driver)
  Sets the graphics driver of this drawing surface.
- virtual void end_current ()
  FLTK calls this each time a surface ceases to be the current drawing surface.
- Fl_Surface_Device (Fl_Graphics_Driver *graphics_driver)
  Constructor that sets the graphics driver to use for the created surface.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()
  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device *new_current)
  Pushes new_current on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()
  The current drawing surface.

Protected Attributes inherited from Fl_Widget_Surface

- int x_offset
  horizontal offset to the origin of graphics coordinates
- int y_offset
  vertical offset to the origin of graphics coordinates

12.27.1 Detailed Description

Supports copying of graphical data to the clipboard.

After creation of an Fl_Copy_Surface object, make it the current drawing surface calling Fl_Surface_Device::push_current(),
and all subsequent graphics requests will be recorded in the clipboard. It's possible to draw widgets (using
Fl_Copy_Surface::draw() ) or to use any of the Drawing functions or the Color & Font functions. Finally, delete the
Fl_Copy_Surface object to load the clipboard with the graphical data.
Fl_Gl_Window 's can be copied to the clipboard as well.

Usage example:

```cpp
Fl_Widget *g = ...; // a widget you want to copy to the clipboard
Fl_Copy_Surface *copy_surf = new Fl_Copy_Surface(g->w(), g->h()); // create an Fl_Copy_Surface object
Fl_Surface_Device::push_current(copy_surf); // direct graphics requests to the clipboard
fl_color(FL_WHITE); fl_rectf(0, 0, g->w(), g->h()); // draw a white background
copy_surf->draw(g); // draw the g widget in the clipboard
Fl_Surface_Device::pop_current(); // direct graphics requests back to their previous destination
delete copy_surf; // after this, the clipboard is loaded
```

Platform details:

- Windows: Transparent RGB images copy without transparency. The graphical data are copied to the clipboard
  in two formats: 1) as an 'enhanced metafile'; 2) as a color bitmap. Applications to which the clipboard content
  can be pasted use the format that suits them best.

- Mac OS: The graphical data are copied to the clipboard (a.k.a. pasteboard) in two 'flavors': 1) in vectorial
  form as PDF data; 2) in bitmap form as a TIFF image. Applications to which the clipboard content is pasted
  can use the flavor that suits them best.

- X11 and Wayland: the graphical data are copied to the clipboard as an image in BMP format.
12.27.2 Constructor & Destructor Documentation

12.27.2.1 Fl_Copy_Surface()

Fl_Copy_Surface::Fl_Copy_Surface (  
  int w,  
  int h )

the constructor

Parameters

| w,h | Width and height of the drawing surface in FLTK units |

12.27.3 Member Function Documentation

12.27.3.1 is_current()

bool Fl_Copy_Surface::is_current ( ) [virtual]

Is this surface the current drawing surface?
Reimplemented from Fl_Surface_Device.

12.27.3.2 origin [1/2]

void Fl_Copy_Surface::origin (  
  int * x,  
  int * y ) [virtual]

Computes the coordinates of the current origin of graphics functions.

Parameters

| out x,y | If non-null, *x and *y are set to the horizontal and vertical coordinates of the graphics origin. |

Reimplemented from Fl_Widget_Surface.

12.27.3.3 origin [2/2]

void Fl_Copy_Surface::origin (  
  int x,  
  int y ) [virtual]

Sets the position of the origin of graphics in the drawable part of the drawing surface.
Arguments should be expressed relatively to the result of a previous printable_rect() call. That is, printable_rect(&w, &h); origin(w/2, 0); sets the graphics origin at the top center of the drawable area. Successive origin() calls don't combine their effects. Origin() calls are not affected by rotate() calls (for classes derived from Fl_Paged_Device).

Parameters

| in x,y | Horizontal and vertical positions in the drawing surface of the desired origin of graphics. |

Reimplemented from Fl_Widget_Surface.

12.27.3.4 printable_rect()

int Fl_Copy_Surface::printable_rect (  
  int * w,  
  int * h ) [virtual]

Computes the width and height of the drawable area of the drawing surface.
Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the
object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

12.27.3.5 set_current()

void Fl_Copy_Surface::set_current {
    void ) [virtual]

Make this surface the current drawing surface.
This surface will receive all future graphics requests. Starting from FLTK 1.4.0, the preferred API to change the current drawing surface is Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current().

Note

It's recommended to use this function only as follows:

- The current drawing surface is the display;
- make current another surface, e.g., an Fl_Printer or an Fl_Image_Surface object, calling set_current() on this object;
- draw to that surface;
- make the display current again with Fl_Display_Device::display_device()-&gt;set_current();. Don't do any other call to set_current() before this one.

Other scenarios of drawing surface changes should be performed via Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current().

Reimplemented from Fl_Surface_Device.

12.27.3.6 translate()

void Fl_Copy_Surface::translate (  
    int x,  
    int y ) [protected], [virtual]

Translates the current graphics origin accounting for the current rotation.
Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.
Reimplemented from Fl_Widget_Surface.

12.27.3.7 untranslate()

void Fl_Copy_Surface::untranslate (  
    void ) [protected], [virtual]

Undoes the effect of a previous translate() call.
Reimplemented from Fl_Widget_Surface.

The documentation for this class was generated from the following files:

- Fl_Copy_Surface.H
- Fl_Copy_Surface.cxx

12.28 Fl_Counter Class Reference

Controls a single floating point value with button (or keyboard) arrows.
#include &lt;Fl_Counter.H&gt;

Inheritance diagram for Fl_Counter:
Public Member Functions

- **FL.Counter** (int X, int Y, int W, int H, const char *L=0)
  Creates a new FL.Counter widget using the given position, size, and label string.
- int **handle** (int) FL_OVERRIDE
  Handles the specified event.
- void **lstep** (double a)
  Sets the increment for the large step buttons.
- double **step** () const
  Returns the increment for normal step buttons.
- void **step** (double a)
  Sets the increment for the normal step buttons.
- void **step** (double a, double b)
  Sets the increments for the normal and large step buttons.
- FL.Color **textcolor** () const
  Gets the font color.
- void **textcolor** (FL.Color s)
  Sets the font color to s.
- FL.Font **textfont** () const
  Gets the text font.
- void **textfont** (FL.Font s)
  Sets the text font to s.
- FL.Fontsize **textsize** () const
  Gets the font size.
- void **textsize** (FL.Fontsize s)
  Sets the font size to s.
- **~FL.Counter** ()
  Destroys the valuator.

Public Member Functions inherited from FL.Valuator

- void **bounds** (double a, double b)
  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)
  Clamps the passed value to the valuator range.
- virtual int **format** (char *)
  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- double **increment** (double, int)
  Adds n times the step value to the passed value.
- double **maximum** () const
  Gets the maximum value for the valuator.
• void **maximum**(double a)
  
  *Sets the maximum value for the valuator.*

• double **minimum** () const
  
  *Gets the minimum value for the valuator.*

• void **minimum**(double a)
  
  *Sets the minimum value for the valuator.*

• void **precision**(int digits)
  
  *Sets the step value to \(1.0 / 10^{\text{digits}}\).*

• void **range**(double a, double b)
  
  *Sets the minimum and maximum values for the valuator.*

• double **round**(double)
  
  *Round the passed value to the nearest step increment.*

• double **step** () const
  
  *Gets or sets the step value.*

• void **step**(double a, int b)
  
  *See double Fl_Valuator::step() const*

• void **step**(double s)
  
  *See double Fl_Valuator::step() const.*

• void **step**(int a)
  
  *See double Fl_Valuator::step() const*

• double **value** () const
  
  *Gets the floating point(double) value.*

• int **value**(double)
  
  *Sets the current value.*

Public Member Functions inherited from Fl_Widget

• void **_clear_fullscreen** ()

• void **_set_fullscreen** ()

• void **activate** ()
  
  *Activates the widget.*

• unsigned int **active** () const
  
  *Returns whether the widget is active.*

• int **active_r** () const
  
  *Returns whether the widget and all of its parents are active.*

• Fl_Align **align** () const
  
  *Gets the label alignment.*

• void **align**(Fl_Align alignment)
  
  *Sets the label alignment.*

• long **argument** () const
  
  *Gets the current user data (long) argument that is passed to the callback function.*

• void **argument**(long v)
  
  *Sets the current user data (long) argument that is passed to the callback function.*

• virtual class Fl_Gl_Window ∗ **as_gl_window** ()
  
  *Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.*

• virtual class Fl_Gl_Window const ∗ **as_gl_window** () const

• virtual Fl_Group ∗ **as_group** ()
  
  *Returns an Fl_Group pointer if this widget is an Fl_Group.*

• virtual Fl_Group const ∗ **as_group** () const

• virtual Fl_Window ∗ **as_window** ()

• virtual Fl_Window const ∗ **as_window** () const
Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const * as_window () const
- void bind_deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
  Checks if the widget value changed since the last callback.
- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()
  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)
  Clears or sets the damage flags.
- void clear_output ()
  Sets a widget to accept input.
- void clear_visible ()
  Hides the widget.
- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
- Fl_Color color () const
  Gets the background color of the widget.
- void color (Fl_Color bg)
  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
- Fl_Color color2 () const
  For back compatibility only.
- void color2 (unsigned a)
  For back compatibility only.
- int contains (const Fl_Widget *w) const
Checks if \( w \) is a child of this widget.

- **void copy_label** (const char *new_label)
  Sets the current label.

- **void copy_tooltip** (const char *text)
  Sets the current tooltip text.

- **uchar damage()** const
  Returns non-zero if \( \text{draw()} \) needs to be called.

- **void damage(uchar c)**
  Sets the damage bits for the widget.

- **void damage(uchar c, int x, int y, int w, int h)**
  Sets the damage bits for an area inside the widget.

- **int damage_resize(int, int, int, int)**
  Internal use only.

- **void deactivate()**
  Deactivates the widget.

- **Fl_Image *deimage()**
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image *deimage() const**
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage(Fl_Image &img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage(Fl_Image *img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound() const**
  Returns whether the inactive image is managed by the widget.

- **void do_callback(Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with default arguments.

- **void do_callback(Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void do_callback(Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void draw_label(int, int, int, int, Fl_Align) const**
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h() const**
  Gets the widget height.

- **virtual void hide()**
  Makes a widget invisible.

- **Fl_Image *image()**
  Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image *image() const**
  Gets the image that is used as part of the widget label when in the active state.

- **void image(Fl_Image &img)**
  Sets the image to use as part of the widget label when in the active state.

- **void image(Fl_Image *img)**
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound() const**
  Returns whether the image is managed by the widget.

- **int inside(const Fl_Widget *wgt) const**
  Checks if this widget is a child of \( wgt \).

- **int is_label_copied() const**
  Returns whether the current label was assigned with \( \text{copy_label()} \).
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
Sets a widget to output only.

- **void** set_visible()
  
  Makes the widget visible.

- **void** set_visible_focus()
  
  Enables keyboard focus navigation with this widget.

- **int** shortcut_label() const
  
  Returns whether the widget's label uses '&' to indicate shortcuts.

- **void** shortcut_label(int value)
  
  Sets whether the widget's label uses '&' to indicate shortcuts.

- **virtual void** show()
  
  Makes a widget visible.

- **void** size(int W, int H)
  
  Changes the size of the widget.

- **int** take_focus()
  
  Gives the widget the keyboard focus.

- **unsigned int** takesevents() const
  
  Returns if the widget is able to take events.

- **int** testShortcut()
  
  Returns true if the widget's label contains the entered '&x' shortcut.

- **const char** * tooltip() const
  
  Gets the current tooltip text.

- **void** tooltip(const char *text)
  
  Sets the current tooltip text.

- **Fl_Window** * top_window() const
  
  Returns a pointer to the top-level window for the widget.

- **Fl_Window** * top_window_offset(int &xoff, int &yoff) const
  
  Finds the x/y offset of the current widget relative to the top-level window.

- **uchar** type() const
  
  Gets the widget type.

- **void** use_accents_menu()
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- **void** * user_data() const
  
  Gets the user data for this widget.

- **void** user_data(Fl_Callback_User_Data*v, bool auto_free)
  
  Sets the user data for this widget.

- **void** user_data(void*v)
  
  Sets the user data for this widget.

- **unsigned int** visible() const
  
  Returns whether a widget is visible.

- **unsigned int** visible_focus() const
  
  Checks whether this widget has a visible focus.

- **void** visible_focus(int v)
  
  Modifies keyboard focus navigation.

- **int** visible_r() const
  
  Returns whether a widget and all its parents are visible.

- **int** w() const
  
  Gets the widget width.

- **Fl_When** when() const
  
  Returns the conditions under which the callback is called.
• void when (uchar i)
    Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
    Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
    Gets the widget position in its window.
• int y () const
    Gets the widget position in its window.
• virtual ~Fl_Widget ()
    Destroys the widget.

Protected Member Functions

• void arrow_widths (int &w1, int &w2)
    Compute sizes (widths) of arrow boxes.
• void draw () FL_OVERRIDE
    Draws the widget.

Protected Member Functions inherited from Fl_Valuator

• Fl_Valuator (int X, int Y, int W, int H, const char ∗L)
    Creates a new Fl_Valuator widget using the given position, size, and label string.
• void handle_drag (double newvalue)
    Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
• void handle_push ()
    Stores the current value in the previous value.
• void handle_release ()
    Called after an FL_WHEN_RELEASE event is received and before the callback.
• int horizontal () const
    Tells if the valuator is an FL_HORIZONTAL one.
• double previous_value () const
    Gets the previous floating point value before an event changed it.
• void set_value (double v)
    Sets the current floating point value.
• double softclamp (double)
    Clamps the value, but accepts v if the previous value is not already out of range.
• virtual void value_damage ()
    Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
    Clears a flag in the flags mask.
• void draw_backdrop () const
    If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const
    Draws the widget box according its box style.
• void draw_box (Fl_Boxtype t, Fl_Color c) const
    Draws a box of type t, of color c at the widget's position and size.
• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
    Draws a box of type t, of color c at the position X,Y and size W,H.
• void draw_focus () const
Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

- void draw_label () const
  Draws the widget's label at the defined label position.

- void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char ∗ label=0L)
  Creates a widget at the given position and size.

- unsigned int flags () const
  Gets the widget flags mask.

- void h (int v)
  Internal use only.

- void set_flag (unsigned int c)
  Sets a flag in the flags mask.

- void w (int v)
  Internal use only.

- void x (int v)
  Internal use only.

- void y (int v)
  Internal use only.

---

**Additional Inherited Members**

**Static Public Member Functions inherited from Fl_Widget**

- static void default_callback (Fl_Widget ∗ widget, void ∗ data)
  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char ∗ t)
  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char ∗ t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

---

**Protected Types inherited from Fl_Widget**

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUSED = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_Tooltip = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCEPTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22, AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
  flags possible values enumeration.

---

Generated by Doxygen
12.28.1 Detailed Description

Controls a single floating point value with button (or keyboard) arrows. Double arrows buttons achieve larger steps than simple arrows.

See also

Fl_Spinner for value input with vertical step arrows.

Figure 12.14 Fl_Counter

The type of an Fl_Counter object can be set using Fl_Widget::type(uchar) to:

- FL_NORMAL_COUNTER: Displays a counter with 4 arrow buttons.
- FL_SIMPLE_COUNTER: Displays a counter with only 2 arrow buttons.

12.28.2 Constructor & Destructor Documentation

12.28.2.1 Fl_Counter()

Fl_Counter::Fl_Counter {
    int X,
    int Y,
    int W,
    int H,
    const char ∗ L = 0
}

Creates a new Fl_Counter widget using the given position, size, and label string. The default type is FL_NORMAL_COUNTER.

Parameters

| in  | X,Y,W,H | position and size of the widget |
| in  | L      | widget label, default is no label |

12.28.3 Member Function Documentation

12.28.3.1 arrow_widths()

void Fl_Counter::arrow_widths ( {
    int & w1,
    int & w2 ) [protected]

Compute sizes (widths) of arrow boxes. This method computes the two sizes of the arrow boxes of Fl_Counter. You can override it in a subclass if you want to draw fancy arrows or change the layout. However, the basic layout is fixed and can't be changed w/o overriding the draw() and handle() methods.

Basic layout:

```
+-----------+-------------+-----+------+
| «     | <     | value     | >   | »     |
+-----------+-------------+-----+------+
```

The returned value w2 should be zero if the counter type() is FL_SIMPLE_COUNTER.

Parameters

| out | w1 | width of single arrow box |

Generated by Doxygen
Parameters

| out | w2 | width of double arrow box |

### 12.28.3.2 draw()

```cpp
def Fl_Counter::draw ( ) [protected], [virtual]

Draws the widget.
```

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own `draw() method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

### 12.28.3.3 handle()

```cpp
def Fl_Counter::handle ( ) [virtual]

Handles the specified event.
```

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class.

In rare cases you may want to return 1 from your `handle()` method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may “ring the bell” or show another visual indication or drop the event silently. In such a case you must not call the `handle()` method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in | event | the kind of event received |

Return values

- 0 if the event was not used or understood
- 1 if the event was used and can be deleted

See also

- `Fl_Event`

Reimplemented from `Fl_Widget`.

### 12.28.3.4 lstep()

```cpp
def Fl_Counter::lstep ( ) [inline]

Sets the increment for the large step buttons.
```

The default value is 1.0.
12.28.3.5 step() [1/2]

void Fl_Counter::step (  
   double a ) [inline]
Sets the increment for the normal step buttons.

Parameters

| in | a | large step increment. |

12.28.3.6 step() [2/2]

void Fl_Counter::step (  
   double a,  
   double b ) [inline]
Sets the increments for the normal and large step buttons.

Parameters

| in | a,b | normal and large step increments. |

The documentation for this class was generated from the following files:

- Fl_Counter.H
- Fl_Counter.cxx

12.29 Fl_Device_Plugin Class Reference

This plugin socket allows the integration of new device drivers for special window or screen types.

```
#include <Fl_Device.H>
```

Inheritance diagram for Fl_Device_Plugin:

```
Fl_Device_Plugin
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
| Fl_Plugin
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
| Fl_Device_Plugin
```

Public Member Functions

- **Fl_Device_Plugin** (const char *pluginName)
  The constructor.
- virtual const char * **klass** ()
  Returns the class name.
- virtual const char * **name** ()=0
  Returns the plugin name.
- virtual int **print** (Fl_Widget *w)=0
  Prints a widget.
- virtual Fl_RGB_Image * **rectangle_capture** (Fl_Widget *widget, int x, int y, int w, int h)=0
  Captures a rectangle of a widget as an image.
Public Member Functions inherited from **Fl_Plugin**

- **Fl_Plugin** (const char *klass, const char *name)
  
  Create a plugin.

- virtual **~Fl_Plugin** ()
  
  Clear the plugin and remove it from the database.

**Static Public Member Functions**

- static **Fl_Device_Plugin** *opengl_plugin ()
  
  Returns the OpenGL plugin.

### 12.29.1 Detailed Description

This plugin socket allows the integration of new device drivers for special window or screen types. This class is not intended for use outside the FLTK library. It is currently used to provide an automated printing service and screen capture for OpenGL windows, if linked with fltk_gl.

### 12.29.2 Member Function Documentation

#### 12.29.2.1 rectangle_capture()

`virtual Fl_RGB_Image * Fl_Device_Plugin::rectangle_capture ( Fl_Widget * widget, int x, int y, int w, int h ) [pure virtual]`

Captures a rectangle of a widget as an image.

**Returns**

The captured pixels as an RGB image.

The documentation for this class was generated from the following files:

- **Fl_Device.H**
- **Fl_Device.cxx**

### 12.30 Fl_Dial Class Reference

The **Fl_Dial** widget provides a circular dial to control a single floating point value.

```cpp
#include <Fl_Dial.H>
```

Inheritance diagram for **Fl_Dial**:

```
Fl_Widget
  ↓
Fl_Valuator
  ↓
Fl_Dial
  ↓
Fl_Fill_Dial Fl_Line_Dial
```

Generated by Doxygen
Public Member Functions

- short angle1 () const
  Sets or gets the angles used for the minimum and maximum values.
- void angle1 (short a)
  See short angle1 () const.
- short angle2 () const
  See short angle1 () const.
- void angle2 (short a)
  See short angle1 () const.
- void angles (short a, short b)
  See short angle1 () const.
- Fl_Dial (int x, int y, int w, int h, const char ∗ l=0)
  Creates a new Fl_Dial widget using the given position, size, and label string.
- int handle (int) FL_OVERRIDE
  Allow subclasses to handle event based on current position and size.

Public Member Functions inherited from Fl_Valuator

- void bounds (double a, double b)
  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double clamp (double)
  Clamps the passed value to the valuator range.
- virtual int format (char ∗)
  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- double increment (double, int)
  Adds n times the step value to the passed value.
- double maximum () const
  Gets the maximum value for the valuator.
- void maximum (double a)
  Sets the maximum value for the valuator.
- double minimum () const
  Gets the minimum value for the valuator.
- void minimum (double a)
  Sets the minimum value for the valuator.
- void precision (int digits)
  Sets the step value to \(1.0 / 10^{\text{digits}}\).
- void range (double a, double b)
  Sets the minimum and maximum values for the valuator.
- double round (double)
  Round the passed value to the nearest step increment.
- double step () const
  Gets or sets the step value.
- void step (double a, int b)
  See double Fl_Valuator::step () const
- void step (double s)
  See double Fl_Valuator::step () const.
- void step (int a)
  See double Fl_Valuator::step () const
- double value () const
  Gets the floating point (double) value.
- int value (double)
  Sets the current value.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen()
- void _set_fullscreen()
- void activate()
  Activates the widget.
- unsigned int active() const
  Returns whether the widget is active.
- int active_r() const
  Returns whether the widget and all of its parents are active.
- Fl_Align align() const
  Gets the label alignment.
- void align(Fl_Align alignment)
  Sets the label alignment.
- long argument() const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument(long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * as_gl_window() const
- virtual Fl_Group * as_group()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * as_group() const
- virtual Fl_Window * as_window()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * as_window() const
- void bind_deimage(Fl_Image * img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage(int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image(Fl_Image * img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image(int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box() const
  Gets the box type of the widget.
- void box(Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback() const
  Gets the current callback function for the widget.
- void callback(Fl_Callback * cb)
  Sets the current callback function for the widget.
- void callback(Fl_Callback * cb, Fl_Callback_User_Data * p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback(Fl_Callback * cb, void *p)
  Sets the current callback function and data for the widget.
- void callback(Fl_Callback0 * cb)
  Sets the current callback function for the widget.
- void callback(Fl_Callback1 * cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed() const
Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget * w) const
  Checks if w is a child of this widget.
• void copy_label (const char * new_label)
  Sets the current label.
• void copy_tooltip (const char * text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image * deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image * deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image *image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image *image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char *label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
Sets whether this widget needs a keyboard.

- **unsigned int output () const**
  
  Returns if a widget is used for output only.

- **Fl_Group * parent () const**
  
  Returns a pointer to the parent widget.

- **void parent (Fl_Group *p)**
  
  Internal use only - "for hacks only".

- **void position (int X, int Y)**
  
  Repositions the window or widget.

- **void redraw ()**
  
  Schedules the drawing of the widget.

- **void redraw_label ()**
  
  Schedules the drawing of the label.

- **virtual void resize (int x, int y, int w, int h)**
  
  Changes the size or position of the widget.

- **Fl_Color selection_color () const**
  
  Gets the selection color.

- **void selection_color (Fl_Color a)**
  
  Sets the selection color.

- **void set_active ()**
  
  Marks the widget as active without sending events or changing focus.

- **void set_changed ()**
  
  Marks the value of the widget as changed.

- **void set_output ()**
  
  Sets a widget to output only.

- **void set_visible ()**
  
  Makes the widget visible.

- **void set_visible_focus ()**
  
  Enables keyboard focus navigation with this widget.

- **int shortcut_label () const**
  
  Returns whether the widget's label uses '&' to indicate shortcuts.

- **void shortcut_label (int value)**
  
  Sets whether the widget's label uses '&' to indicate shortcuts.

- **virtual void show ()**
  
  Makes a widget visible.

- **void size (int W, int H)**
  
  Changes the size of the widget.

- **int take_focus ()**
  
  Gives the widget the keyboard focus.

- **unsigned int takesevents () const**
  
  Returns if the widget is able to take events.

- **int test_shortcut ()**
  
  Returns true if the widget's label contains the entered '&x' shortcut.

- **const char * tooltip () const**
  
  Gets the current tooltip text.

- **void tooltip (const char *text)**
  
  Sets the current tooltip text.

- **Fl_Window * top_window () const**
  
  Returns a pointer to the top-level window for the widget.

- **Fl_Window * top_window_offset (int &xoff, int &yoff) const**
  
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void *user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window *window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.

Protected Member Functions

• void draw () FL_OVERRIDE
  Draws dial at current position and size.
• void draw (int X, int Y, int W, int H)
  Draws dial at given position and size.
• int handle (int event, int X, int Y, int W, int H)
  Allows subclasses to handle event based on given position and size.

Protected Member Functions inherited from Fl_Valuator

• Fl_Valuator (int X, int Y, int W, int H, const char *L)
  Creates a new Fl_Valuator widget using the given position, size, and label string.
• void handle_drag (double newvalue)
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
• void handle_push ()
  Stores the current value in the previous value.
• void **handle_release** ()
  
  Called after an FL_WHEN_RELEASE event is received and before the callback.

• int **horizontal** () const
  
  Tells if the valuator is an FL_HORIZONTAL one.

• double **previous_value** () const
  
  Gets the previous floating point value before an event changed it.

• void **set_value** (double v)
  
  Sets the current floating point value.

• double **softclamp** (double)
  
  Clamps the value, but accepts v if the previous value is not already out of range.

• virtual void **value_damage** ()
  
  Asks for partial redraw.

**Protected Member Functions inherited from Fl_Widget**

• void **clear_flag** (unsigned int c)
  
  Clears a flag in the flags mask.

• void **draw_backdrop** () const
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• void **draw_box** () const
  
  Draws the widget box according its box style.

• void **draw_box** (Fl_Boxtype t, Fl_Color c) const
  
  Draws a box of type t, of color c at the widget's position and size.

• void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type t, of color c at the position X,Y and size W,H.

• void **draw_focus** () const
  
  Draws a focus rectangle around the widget.

• void **draw_focus** (Fl_Boxtype t, int X, int Y, int W, int H) const
  
  Draws a focus rectangle around the widget.

• void **draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.

• void **draw_label** () const
  
  Draws the widget's label at the defined label position.

• void **draw_label** (int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.

• Fl_Widget **(int x, int y, int w, int h, const char *label=0L)**
  
  Creates a widget at the given position and size.

• unsigned int **flags** () const
  
  Gets the widget flags mask.

• void **h** (int v)
  
  Internal use only.

• void **set_flag** (unsigned int c)
  
  Sets a flag in the flags mask.

• void **w** (int v)
  
  Internal use only.

• void **x** (int v)
  
  Internal use only.

• void **y** (int v)
  
  Internal use only.
Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget *widget, void *data)
  
  The default callback for all widgets that don’t set a callback.

• static unsigned int labelShortcut (const char *t)
  
  Returns the Unicode value of the ‘&x’ shortcut in a given text.

• static int testShortcut (const char *t, const bool require_alt=false)
  
  Returns true if the given text t contains the entered ‘&x’ shortcut.

Protected Types inherited from Fl_Widget

• enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9,copied_label = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCEPTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

flags possible values enumeration.

12.30.1 Detailed Description

The Fl.Dial widget provides a circular dial to control a single floating point value.

Use type() to set the type of the dial to:

• FL_NORMAL_DIAL - Draws a normal dial with a knob.

• FL_LINE_DIAL - Draws a dial with a line.

• FL_FILL_DIAL - Draws a dial with a filled arc.

12.30.2 Constructor & Destructor Documentation

12.30.2.1 Fl.Dial()

Fl.Dial::Fl.Dial ( 
    int X,
    int Y,
    int W,
    int H,
    const char * l = 0 )

Creates a new Fl.Dial widget using the given position, size, and label string.
The default type is FL_NORMAL_DIAL.
12.30.3 Member Function Documentation

12.30.3.1 angle1()

short Fl_Dial::angle1 ( ) const [inline]
Sets Or gets the angles used for the minimum and maximum values.
The default values are 45 and 315 (0 degrees is straight down and the angles progress clockwise). Normally angle1
is less than angle2, but if you reverse them the dial moves counter-clockwise.

12.30.3.2 draw() [1/2]

void Fl_Dial::draw ( void ) [protected], [virtual]
Draws dial at current position and size.
Implements Fl_Widget.

12.30.3.3 draw() [2/2]

void Fl_Dial::draw ( int X, int Y, int W, int H ) [protected]
Draws dial at given position and size.

Parameters

| in | X,Y,W,H | position and size |

12.30.3.4 handle() [1/2]

int Fl_Dial::handle ( int event, int X, int Y, int W, int H ) [protected]
Allows subclasses to handle event based on given position and size.

Parameters

| in | event,X,Y,W,H | event to handle, related position and size. |

12.30.3.5 handle() [2/2]

int Fl_Dial::handle ( int e ) [virtual]
Allow subclasses to handle event based on current position and size.
Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

- Fl_Dial.H
- Fl_Dial.cxx

12.31 Fl_Display_Device Class Reference

The computer’s display.


#include <Fl_Device.H>

Inheritance diagram for Fl_Display_Device:

```
Fl_Surface_Device
    ^
    | Fl_Display_Device
```

Static Public Member Functions

- static Fl_Display_Device * display_device ()
  "Returns a pointer to the unique display device."

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()
  "Removes the top element from the current drawing surface stack, and makes the new top element current."
- static void push_current (Fl_Surface_Device * new_current)
  "Pushes new_current on top of the stack of current drawing surfaces, and makes it current."
- static Fl_Surface_Device * surface ()
  "The current drawing surface."

Additional Inherited Members

Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver * driver ()
  "Returns the graphics driver of this drawing surface."
- virtual bool is_current ()
  "Is this surface the current drawing surface?"
- virtual void set_current (void)
  "Make this surface the current drawing surface."
- virtual ~Fl_Surface_Device ()
  "The destructor."

Protected Member Functions inherited from Fl_Surface_Device

- void driver (Fl_Graphics_Driver * graphics_driver)
  "Sets the graphics driver of this drawing surface."
- virtual void end_current ()
  "FLTK calls this each time a surface ceases to be the current drawing surface."
- Fl_Surface_Device (Fl_Graphics_Driver * graphics_driver)
  "Constructor that sets the graphics driver to use for the created surface."

### 12.31.1 Detailed Description

The computer's display.
When FLTK begins to access the computer's display, it creates an object of class Fl_Display_Device and makes it the current drawing surface.
The documentation for this class was generated from the following files:

- Fl_Device.H
- Fl_Device.cxx
12.32 Fl_Double_Window Class Reference

The Fl_Double_Window provides a double-buffered window.
#include <Fl_Double_Window.H>

Inheritance diagram for Fl_Double_Window:

```
+-----------------------+
| Fl_Widget             |
| (virtual)             |
+-----------------------+    +-----------------------+
| Fl_Group              |    | Fl_Window             |
| (virtual)             |    +-----------------------+    +-----------------------+
| Fl_Double_Window      |    | Fl_Cairo_Window       |
| (virtual)             |    | Fl_Overlay_Window     |
+-----------------------+    +-----------------------+    +-----------------------+
```

Public Member Functions

- Fl_Double_Window * as_double_window () FL_OVERRIDE
  Return non-null if this is an Fl_Overlay_Window object.
- Fl_Double_Window (int W, int H, const char *l=0)
  Creates a new Fl_Double_Window widget using the given position, size, and label (title) string.
- Fl_Double_Window (int X, int Y, int W, int H, const char *l=0)
  See Fl_Double_Window::Fl_Double_Window(int w, int h, const char *label = 0)
- void flush () FL_OVERRIDE
  Forcing the window to be drawn, this window is also made current and calls draw().
- void hide () FL_OVERRIDE
  Makes a widget invisible.
- void resize (int, int, int) FL_OVERRIDE
  Changes the size or position of the widget.
- void show () FL_OVERRIDE
  Makes a widget visible.
- void show (int a, char **b)
  Same as Fl_Window::show(int a, char **b)
- ~Fl_Double_Window ()
  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Window

- virtual class Fl_Overlay_Window * as_overlay_window ()
  Return non-null if this is an Fl_Overlay_Window object.
- Fl_Window const * as_window () const FL_OVERRIDE
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int border () const
  Returns whether the window possesses a border.
- void border (int b)
  Sets whether or not the window manager border is around the window.
- void clear_border ()
  Fast inline function to turn the window manager border off.
• void clear_modal_states ()

Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.

• void copy_label (const char *a)

Sets the window titlebar label to a copy of a character string.

• void cursor (const Fl_RGB_Image *, int, int)

Changes the cursor for this window using the provided image as cursor's shape.

• void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

For back compatibility only.

• void cursor (Fl_Cursor)

Changes the cursor for this window.

• int decorated_h () const

Returns the window height including any window title bar and any frame added by the window manager.

• int decorated_w () const

Returns the window width including any frame added by the window manager.

• void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

For back compatibility only.

• void default_cursor (Fl_Cursor)

Sets the default window cursor.

• void draw_backdrop ()

Draw the background image if one is set and is aligned inside.

• Fl_Window (int w, int h, const char *title=0)

Creates a window from the given width w, height h, and title.

• Fl_Window (int x, int y, int w, int h, const char *title=0)

Creates a window from the given position (x, y), size (w, h) and title.

• void free_position ()

Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.

• void fullscreen ()

Makes the window completely fill one or more screens, without any window manager border visible.

• unsigned int fullscreen_active () const

Returns non zero if FULLSCREEN flag is set, 0 otherwise.

• void fullscreen_off ()

Turns off any side effects of fullscreen()

• void fullscreen_off (int X, int Y, int W, int H)

Turns off any side effects of fullscreen() and does resize(x,y,w,h).

• void fullscreen_screens (int top, int bottom, int left, int right)

Sets which screens should be used when this window is in fullscreen mode.

• int handle (int) FL_OVERRIDE

Handles the specified event.

• void hide () FL_OVERRIDE

Removes the window from the screen.

• void hotspot (const Fl_Widget &p, int offscreen=0)

See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

• void hotspot (const Fl_Widget *, int offscreen=0)

See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

• void hotspot (int x, int y, int offscreen=0)

Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.

• const void * icon () const

Gets the current icon window target dependent data.

• void icon (const Fl_RGB_Image *)

Generated by Doxygen
Sets or resets a single window icon.

- **void icon (const void *ic)**
  
  Platform-specific method to set the window icon usable on Windows and X11 only.

- **void iconize ()**
  
  Iconifies the window.

- **const char * iconlabel () const**
  
  See void Fl_Window::iconlabel(const char *)

- **void iconlabel (const char *)**
  
  Sets the icon label.

- **void icons (const Fl_RGB_Image *[], int)**
  
  Sets the window icons.

- **void icons (HICON big_icon, HICON small_icon)**
  
  Sets the window icons using HICON handles (Windows platform only).

- **const char * label () const**
  
  See void Fl_Window::label(const char *)

- **void label (const char *)**
  
  Sets the window title bar label.

- **void label (const char *label, const char *iconlabel)**
  
  Sets the icon label.

- **void make_current ()**
  
  Sets things up so that the drawing functions in `<FL/fl_draw.H>` will go into this window.

- **void maximize ()**
  
  Maximizes a top-level window to its current screen.

- **unsigned int maximize_active () const**
  
  Returns whether the window is currently maximized.

- **unsigned int menu_window () const**
  
  Returns true if this window is a menu window.

- **unsigned int modal () const**
  
  Returns true if this window is modal.

- **unsigned int non_modal () const**
  
  Returns true if this window is modal or non-modal.

- **fl_uintptr_t os_id ()**
  
  Returns a platform-specific identification of a shown window, or 0 if not shown.

- **unsigned int override () const**
  
  Returns non zero if OVERRIDE flag is set, 0 otherwise.

- **void resize (int X, int Y, int W, int H)**
  
  Changes the size and position of the window.

- **int screen_num ()**
  
  The number of the screen containing the mapped window.

- **void screen_num (int screen_num)**
  
  Set the number of the screen where to map the window.

- **void set_menu_window ()**
  
  Marks the window as a menu window.

- **void set_modal ()**
  
  A “modal” window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the “transient for” property).

- **void set_non_modal ()**
  
  A “non-modal” window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.
• void **set_override** ()
  
  Activates the flags NOBORDER|OVERRIDE.

• void **set_tooltip_window** ()
  
  Marks the window as a tooltip window.

• const Fl_Image ∗ **shape** ()
  
  Returns the image controlling the window shape or NULL.

• void shape (const Fl_Image &b)
  
  Set the window's shape with an Fl_Image.

• void shape (const Fl_Image ∗img)
  
  Assigns a non-rectangular shape to the window.

• void **show** () FL_OVERRIDE
  
  Puts the window on the screen.

• void **show** (int argc, char **argv)
  
  Puts the window on the screen with show() and parses command-line arguments.

• int **shown** ()
  
  Returns non-zero if show() has been called (but not hide() ).

• void **size_range** (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)
  
  Sets the allowable range the user can resize this window to.

• unsigned int **tooltip_window** () const
  
  Returns true if this window is a tooltip window.

• void un_maximize () const
  
  Returns a previously maximized top-level window to its previous size.

• void **wait_for_expose** ()
  
  Waits for the window to be displayed after calling show().

• int x_root () const
  
  Gets the x position of the window on the screen.

• const char ∗ xclass () const
  
  Returns the xclass for this window, or a default.

• void xclass (const char ∗c)
  
  Sets the xclass for this window.

• int y_root () const
  
  Gets the y position of the window on the screen.

• virtual ~FL_Window ()
  
  The destructor also deletes all the children.

**Public Member Functions inherited from Fl_Group**

• Fl_Widget ∗& **_ddfdesign_kludge** ()
  
  This is for forms compatibility only.

• void **add** (Fl_Widget &)
  
  The widget is removed from its current group (if any) and then added to the end of this group.

• void **add** (Fl_Widget ∗o)
  
  See void Fl_Group::add(Fl_Widget &w)

• void **add resizable** (Fl_Widget ∗o)
  
  Adds a widget to the group and makes it the resizable widget.

• Fl_Widget ∗const ∗ **array** () const
  
  Returns a pointer to the array of children.

• Fl_Group ∗const ∗ as_group () const FL_OVERRIDE

• Fl_Group ∗ as_group () FL_OVERRIDE
  
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• void **begin** ()
Sets the current group so you can build the widget tree by just constructing the widgets.

- `Fl_Widget * child (int n) const`  
  Returns array()[n].

- `int children () const`  
  Returns how many child widgets the group has.

- `void clear ()`  
  Deletes all child widgets from memory recursively.

- `unsigned int clip_children ()`  
  Returns the current clipping mode.

- `void clip_children (int c)`  
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- `virtual int delete_child (int n)`  
  Removes the widget at index from the group and deletes it.

- `void end ()`  
  Exactly the same as current(this->parent()).

- `int find (const Fl_Widget &o) const`  
  See int Fl_Group::find(const Fl_Widget *w) const.

- `int find (const Fl_Widget *) const`  
  Searches the child array for the widget and returns the index.

- `Fl_Group (int, int, int, int, const char *)=0)`  
  Creates a new Fl_Group widget using the given position, size, and label string.

- `void focus (Fl_Widget *W)`  
  This is for forms compatibility only.

- `void forms_end ()`  
  This does insert(w, find(before)).

- `void init_sizes ()`  
  Resets the internal array of widget sizes and positions.

- `void insert (Fl_Widget &, int i)`  
  The widget is removed from its current group (if any) and then inserted into this group.

- `void insert (Fl_Widget &o, Fl_Widget *before)`  
  This does insert(w, find(before)).

- `void remove (Fl_Widget &)`  
  Removes a widget from the group but does not delete it.

- `void remove (Fl_Widget *o)`  
  Removes the widget o from the group.

- `void remove (int index)`  
  Removes the widget at index from the group but does not delete it.

- `Fl_Widget * resizable () const`  
  Returns the group's resizable widget.

- `void resizable (Fl_Widget &o)`  
  Sets the group's resizable widget.

- `void resizable (Fl_Widget *o)`  
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- `virtual ~Fl_Group ()`  
  The destructor also deletes all the children.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ as_gl_window () const
- void bind_deimage (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
  Checks if the widget value changed since the last callback.
- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget ∗w) const
  Checks if w is a child of this widget.
• void copy_label (const char ∗new_label)
  Sets the current label.
• void copy_tooltip (const char ∗text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Imaug = deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Imaug = deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Imaug &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Imaug =img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Determines whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reasong reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget ∗+widget, long arg, Fl_Callback_Reasong reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget ∗+widget, void ∗+arg=0, Fl_Callback_Reasong reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, Fl_Align) const
Draws the label in an arbitrary bounding box with an arbitrary alignment.

- int `h () const`
  
  Gets the widget height.

- Fl_Image * image ()
  
  Gets the image that is used as part of the widget label when in the active state.

- const Fl_Image * image () const
  
  Gets the image that is used as part of the widget label when in the active state.

- void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the active state.

- void image (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the active state.

- int `image_bound () const`
  
  Returns whether the image is managed by the widget.

- int `inside (const Fl_Widget *wgt) const`
  
  Checks if this widget is a child of `wgt`.

- int `is_label_copied () const`
  
  Returns whether the current label was assigned with `copy_label()`.

- const char * `label () const`
  
  Gets the current label text.

- void `label (const char *text)`
  
  Sets the current label pointer.

- void `label (Fl_Labeltype a, const char *b)`
  
  Shortcut to set the label text and type in one call.

- Fl_Color `labelcolor () const`
  
  Gets the label color.

- void `labelcolor (Fl_Color c)`
  
  Sets the label color.

- Fl_Font `labelfont () const`
  
  Gets the font to use.

- void `labelfont (Fl_Font f)`
  
  Sets the font to use.

- Fl_Fontsize `labelsiz e () const`
  
  Gets the font size in pixels.

- void `labelsiz e (Fl_Fontsize pix)`
  
  Sets the font size in pixels.

- Fl_Labeltype `labeltype () const`
  
  Gets the label type.

- void `labeltype (Fl_Labeltype a)`
  
  Sets the label type.

- void `measure_label (int &ww, int &hh) const`
  
  Sets width `ww` and height `hh` accordingly with the label size.

- bool `needs_keyboard () const`
  
  Returns whether this widget needs a keyboard.

- void `needs_keyboard (bool needs)`
  
  Sets whether this widget needs a keyboard.

- unsigned int `output () const`
  
  Returns if a widget is used for output only.

- Fl_Group * `parent () const`
  
  Returns a pointer to the parent widget.

- void `parent (Fl_Group *p)`
  
  Internal use only - “for hacks only”.

Generated by Doxygen
• void **position**(int X, int Y)
  Repositions the window or widget.

• void **redraw**()
  Schedules the drawing of the widget.

• void **redraw_label**()
  Schedules the drawing of the label.

• **Fl_Color** **selection_color** () const
  Gets the selection color.

• void **selection_color**(Fl_Color a)
  Sets the selection color.

• void **set_active**()
  Marks the widget as active without sending events or changing focus.

• void **set_changed**()
  Marks the value of the widget as changed.

• void **set_output**()
  Sets a widget to output only.

• void **set_visible**()
  Makes the widget visible.

• void **set_visible_focus**()
  Enables keyboard focus navigation with this widget.

• int **shortcut_label** () const
  Returns whether the widget's label uses '&' to indicate shortcuts.

• void **shortcut_label** (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

• void **size** (int W, int H)
  Changes the size of the widget.

• int **take_focus**()
  Gives the widget the keyboard focus.

• unsigned int **takesevents** () const
  Returns if the widget is able to take events.

• int **test_shortcut**()
  Returns true if the widget's label contains the entered '&x' shortcut.

• const char **∗** **tooltip** () const
  Gets the current tooltip text.

• void **tooltip**(const char **∗** text)
  Sets the current tooltip text.

• **Fl_Window** **∗** **top_window** () const
  Returns a pointer to the top-level window for the widget.

• **Fl_Window** **∗** **top_window_offset** (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar **type** () const
  Gets the widget type.

• void **type** (uchar t)
  Sets the widget type.

• int **use_accents_menu** ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void **∗** **user_data** () const
  Gets the user data for this widget.

• void **user_data**(Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.

• void **user_data** (void ∗v)
Sets the user data for this widget.

- **unsigned int visible () const**
  Returns whether a widget is visible.

- **unsigned int visible_focus () const**
  Checks whether this widget has a visible focus.

- **void visible_focus (int v)**
  Modifies keyboard focus navigation.

- **int visible_r () const**
  Returns whether a widget and all its parents are visible.

- **int w () const**
  Gets the widget width.

- **Fl_When when () const**
  Returns the conditions under which the callback is called.

- **void when (uchar i)**
  Sets the flags used to decide when a callback is called.

- **Fl_Window * window () const**
  Returns a pointer to the nearest parent window up the widget hierarchy.

- **int x () const**
  Gets the widget position in its window.

- **int y () const**
  Gets the widget position in its window.

- **virtual ~Fl_Widget ()**
  Destroys the widget.

### Additional Inherited Members

**Public Types inherited from Fl_Window**

- typedef struct HICON__ * HICON

**Static Public Member Functions inherited from Fl_Window**

- **static Fl_Window * current ()**
  Returns the last window that was made current.

- **static void default_callback (Fl_Window *, void *)**
  Back compatibility: Sets the default callback v for win to call on close event.

- **static void default_icon (const Fl_RGB_Image *)**
  Sets a single default window icon.

- **static void default_icons (const Fl_RGB_Image *[], int)**
  Sets the default window icons.

- **static void default_icons (HICON big_icon, HICON small_icon)**
  Sets the default window icons (Windows platform only).

- **static const char * default_xclass ()**
  Returns the default xclass.

- **static void default_xclass (const char *)**
  Sets the default window xclass.

- **static bool is_a_rescale ()**
  Returns true when a window is being rescaled.

- **static char show_next_window_iconic ()**
  Returns the static flag whether the next window should be opened iconified.

- **static void show_next_window_iconic (char stat)**
  Sets a static flag whether the next window should be opened iconified.
Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()
  Returns the currently active group.
- static void current (Fl_Group *g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Window

- void default_size_range ()
  Protected method to calculate the default size range of a window.
- void draw () FL_OVERRIDE
  Draws the widget.
- int force_position () const
  Returns the internal state of the window's FORCE_POSITION flag.
- void force_position (int force)
  Sets an internal flag that tells FLTK and the window manager to honor position requests.
- void free_icons ()
  Deletes all icons previously attached to the window.
- int is_resizable ()
  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.
- void draw_children ()
Draws all children of the group.

- void `draw_outside_label` (const `Fl_Widget` &widget) const

  Parents normally call this to draw outside labels of child widgets.

- virtual int `on_insert` (`Fl_Widget` *, int)

  Allow derived groups to act when a widget is added as a child.

- virtual int `on_move` (int, int)

  Allow derived groups to act when a widget is moved within the group.

- virtual void `on_remove` (int)

  Allow derived groups to act when a child widget is removed from the group.

- int * `sizes` ()

  Returns the internal array of widget sizes and positions.

- void `update_child` (`Fl_Widget` &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from `Fl_Widget`

- void `clear_flag` (unsigned int c)

  Clears a flag in the flags mask.

- void `draw_backdrop` () const

  If `FL_ALIGN_IMAGE_BACKDROP` is set, the image or deimage will be drawn.

- void `draw_box` () const

  Draws the widget box according its box style.

- void `draw_box` (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.

- void `draw_box` (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.

- void `draw_focus` () const

  Draws a focus rectangle around the widget.

- void `draw_focus` (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.

- void `draw_focus` (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.

- void `draw_label` () const

  Draws the widget's label at the defined label position.

- void `draw_label` (int, int, int, int) const

  Draws the label in an arbitrary bounding box.

- `Fl_Widget` (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.

- unsigned int `flags` () const

  Gets the widget flags mask.

- void `h` (int v)

  Internal use only.

- void `set_flag` (unsigned int c)

  Sets a flag in the flags mask.

- void `w` (int v)

  Internal use only.

- void `x` (int v)

  Internal use only.

- void `y` (int v)

  Internal use only.
Static Protected Attributes inherited from Fl_Window

- static Fl_Window * current_
  Stores the last window that was made current.

12.32.1 Detailed Description

The Fl_Double_Window provides a double-buffered window. It will draw the window data into an off-screen pixmap, and then copy it to the on-screen window.

12.32.2 Constructor & Destructor Documentation

12.32.2.1 ~Fl_Double_Window()

Fl_Double_Window::~Fl_Double_Window()

The destructor also deletes all the children.
This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code.

12.32.3 Member Function Documentation

12.32.3.1 as_double_window()

Fl_Double_Window * Fl_Double_Window::as_double_window() [inline], [virtual]

Return non-null if this is an Fl_Overlay_Window object.
Reimplemented from Fl_Window.

12.32.3.2 flush()

void Fl_Double_Window::flush() [virtual]

Forces the window to be drawn, this window is also made current and calls draw().
Reimplemented from Fl_Window.
Reimplemented in Fl_Overlay_Window.

12.32.3.3 hide()

void Fl_Double_Window::hide() [virtual]

Makes a widget invisible.
See also

  show(), visible(), visible_r()

Reimplemented from Fl_Widget.
Reimplemented in Fl_Overlay_Window.

12.32.3.4 resize()

void Fl_Double_Window::resize(
  int x,
  int y,
  int w,
  int h) [virtual]

Changes the size or position of the widget.
This is a virtual function so that the widget may implement its own handling of resizing. The default version does not call the redraw() method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position.
Some window managers under X11 call resize() a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations.
position(X, Y) is a shortcut for resize(X, Y, w(), h()), and size(W, H) is a shortcut for resize(x(), y(), W, H).
12.33 Fl_End Class Reference

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y</td>
<td>new position relative to the parent window</td>
</tr>
<tr>
<td>in</td>
<td>w,h</td>
<td>new size</td>
</tr>
</tbody>
</table>

See also

- position(int,int), size(int,int)

Reimplemented from Fl_Widget.
Reimplemented in Fl_Overlay_Window.

12.32.3.5 show()

void Fl_Double_Window::show ( ) [virtual]

Makes a widget visible.
An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other events like FL_SHOW.
The visible() method returns true if the widget is set to be visible. The visible_r() method returns true if the widget and all of its parents are visible. A widget is only visible if visible() is true on it and all of its parents.
Changing it will send FL_SHOW or FL_HIDE events to the widget. Do not change it if the parent is not visible, as this will send false FL_SHOW or FL_HIDE events to the widget.
redraw() is called if necessary on this or the parent.

See also

- hide(), visible(), visible_r()

Reimplemented from Fl_Widget.
Reimplemented in Fl_Overlay_Window.
The documentation for this class was generated from the following files:

- Fl_Double_Window.H
- Fl_Double_Window.cxx

12.33 Fl_End Class Reference

This is a dummy class that allows you to end a Fl_Group in a constructor list of a class:
#include <Fl_Group.H>

Public Member Functions

- Fl_End ()
  
  All it does is calling Fl_Group::current()->end()

12.33.1 Detailed Description

This is a dummy class that allows you to end a Fl_Group in a constructor list of a class:
class MyClass {
  Fl_Group group;
  Fl_Button button_in_group;
  Fl_End end;
  Fl_Button button_outside_group;
  MyClass();
};
MyClass::MyClass() :
group(10,10,100,100),
button_in_group(20,20,60,30),
end(),
button_outside_group(10,120,60,30) {
  [.ctor code..]
}

The documentation for this class was generated from the following file:

- Fl_Group.H

Generated by Doxygen
12.34  Fl_EPS_File_Surface Class Reference

Encapsulated PostScript drawing surface.
#include <Fl_PostScript.H>
Inheritance diagram for Fl_EPS_File_Surface:

```
Fl_Surface_Device
   |          |
   |          |
   Fl_Widget_Surface
   |          |
   |          |
Fl_EPS_File_Surface
```

Public Member Functions

- int close ()
  Completes all EPS output.
- FILE * file ()
  Returns the underlying FILE pointer.
- Fl_EPS_File_Surface (int width, int height, FILE *eps_output, Fl_Color background=FL_WHITE, Fl_PostScript_Close_Command closef=NULL)
  Constructor.
- void origin (int *px, int *py) FL_OVERRIDE
  Computes the coordinates of the current origin of graphics functions.
- void origin (int x, int y) FL_OVERRIDE
  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- int printable_rect (int *w, int *h) FL_OVERRIDE
  Computes the width and height of the drawable area of the drawing surface.
- void translate (int x, int y) FL_OVERRIDE
  Translates the current graphics origin accounting for the current rotation.
- void untranslate () FL_OVERRIDE
  Undoes the effect of a previous translate() call.
- ~Fl_EPS_File_Surface ()
  Destructor.

Public Member Functions inherited from Fl_Widget_Surface

- void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)
  Draws the widget on the drawing surface.
- void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)
  Draws a window with its title bar and frame if any.
- void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)
  Draws a rectangular part of an on-screen window.

Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver * driver ()
  Returns the graphics driver of this drawing surface.
- virtual bool is_current ()
  Is this surface the current drawing surface?
- virtual void set_current (void)
  Make this surface the current drawing surface.
- virtual ~Fl_Surface_Device ()
  The destructor.
### Protected Member Functions

- **Fl_PostScript_Graphics_Driver * driver ()**
  
  Returns the PostScript driver of this drawing surface.

### Protected Member Functions inherited from Fl_Widget_Surface

- **Fl_Widget_Surface (Fl_Graphics_Driver *d)**
  
  The constructor.

### Protected Member Functions inherited from Fl_Surface_Device

- **void driver (Fl_Graphics_Driver *graphics_driver)**
  
  Sets the graphics driver of this drawing surface.
- **virtual void end_current ()**
  
  FLTK calls this each time a surface ceases to be the current drawing surface.
- **Fl_Surface_Device (Fl_Graphics_Driver *graphics_driver)**
  
  Constructor that sets the graphics driver to use for the created surface.

### Additional Inherited Members

### Static Public Member Functions inherited from Fl_Surface_Device

- **static Fl_Surface_Device * pop_current ()**
  
  Removes the top element from the current drawing surface stack, and makes the new top element current.
- **static void push_current (Fl_Surface_Device *new_current)**
  
  Pushes new_current on top of the stack of current drawing surfaces, and makes it current.
- **static Fl_Surface_Device * surface ()**
  
  The current drawing surface.

### Protected Attributes inherited from Fl_Widget_Surface

- **int x_offset**
  
  Horizontal offset to the origin of graphics coordinates
- **int y_offset**
  
  Vertical offset to the origin of graphics coordinates

### 12.34.1 Detailed Description

Encapsulated PostScript drawing surface.

This drawing surface allows to store any FLTK graphics in vectorial form in an “Encapsulated PostScript” file.

**Usage example:**

```cpp
Fl_Window *win = ...// Window to draw to an .eps file
int ww = win->decorated_w();
int wh = win->decorated_h();
FILE *eps = fl_fopen("/path/to/mywindow.eps", "w");
if (eps) {
    Fl_EPS_File_Surface *surface = new Fl_EPS_File_Surface(ww, wh, eps, win->color());
    Fl_Surface_Device::push_current(surface);
    surface->draw_decorated_window(win);
    Fl_Surface_Device::pop_current();
    delete surface; // the .eps file is not complete until the destructor was run
}
```

Generated by Doxygen
12.34.2 Constructor & Destructor Documentation

12.34.2.1 Fl_EPS_File_Surface()

Fl_EPS_File_Surface::Fl_EPS_File_Surface (  
  int width,  
  int height,  
  FILE * eps_output,  
  Fl_Color background = FL_WHITE,  
  Fl_PostScript_Close_Command closef = NULL )

Constructor.

Parameters

| width, height | Width and height of the EPS drawing area |
| eps_output   | A writable FILE pointer where the Encapsulated PostScript data will be sent |
| background   | Color expected to cover the background of the EPS drawing area. This parameter affects only the drawing of transparent Fl_RGB_Imagel objects: transparent areas of RGB images are blended with the background color. Under the X11 + pango platform, transparent RGB images are correctly blended to their background, thus this parameter has no effect. |
| closef       | If not NULL, the destructor or close() will call closef(eps_output) after all EPS data has been sent. If NULL, fclose(eps_output) is called instead. This allows to close the FILE pointer by, e.g., pclose, or, using a function such as "int keep_open(FILE*)(return 0;)", to keep it open after completion of all output to eps_output. Function closef should return non zero to indicate an error. |

12.34.2.2 ~Fl_EPS_File_Surface()

Fl_EPS_File_Surface::~Fl_EPS_File_Surface ( )

Destructor.

By default, the destructor closes with function fclose() the underlying FILE. See the constructor for how to close it differently or to keep it open. Use close() before object destruction to receive the status code of output operations. If close() is not used and if EPS output results in error, the destructor displays an alert message with fl_alert().

12.34.3 Member Function Documentation

12.34.3.1 close()

int Fl_EPS_File_Surface::close ( )

Completes all EPS output.

The only operation possible with the Fl_EPS_File_Surface object after calling close() is its destruction.

Returns

The status code of output operations to the FILE object. 0 indicates success.

12.34.3.2 origin() [1/2]

void Fl_EPS_File_Surface::origin (  
  int * x,  
  int * y ) [virtual]

Computes the coordinates of the current origin of graphics functions.

Parameters

| out x,y | If non-null, +x and +y are set to the horizontal and vertical coordinates of the graphics origin. |

Reimplemented from Fl_Widget_Surface.
12.34.3.3 \texttt{origin()} [2/2]

\begin{verbatim}
void Fl_EPS_File_Surface::origin (  
    int \textit{x},  
    int \textit{y} ) \text{ [virtual]}
\end{verbatim}

Sets the position of the origin of graphics in the drawable part of the drawing surface. Arguments should be expressed relatively to the result of a previous \texttt{printable_rect()} call. That is, \texttt{printable_rect(&w, &h); origin(w/2, 0);} sets the graphics origin at the top center of the drawable area. Successive \texttt{origin()} calls don't combine their effects. \texttt{origin()} calls are not affected by \texttt{rotate()} calls (for classes derived from \texttt{Fl_Paged_Device}).

\textbf{Parameters}

\begin{tabular}{c|c}
\hline
\textit{x,y} & Horizontal and vertical positions in the drawing surface of the desired origin of graphics. \\
\hline
\end{tabular}

Reimplemented from \texttt{Fl_Widget_Surface}.

12.34.3.4 \texttt{printable_rect()}

\begin{verbatim}
int Fl_EPS_File_Surface::printable_rect (  
    int \* \textit{w},  
    int \* \textit{h} ) \text{ [virtual]}
\end{verbatim}

Computes the width and height of the drawable area of the drawing surface. Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to \texttt{origin()}. If the object is derived from class \texttt{Fl_Paged_Device}, values account for the user-selected paper type and print orientation and are changed by \texttt{scale()} calls.

\textbf{Returns}

\begin{itemize}
\item 0 if OK, non-zero if any error
\end{itemize}

Reimplemented from \texttt{Fl_Widget_Surface}.

12.34.3.5 \texttt{translate()}

\begin{verbatim}
void Fl_EPS_File_Surface::translate (  
    int \textit{x},  
    int \textit{y} ) \text{ [virtual]}
\end{verbatim}

Translates the current graphics origin accounting for the current rotation. Each \texttt{translate()} call must be matched by an \texttt{untranslate()} call. Successive \texttt{translate()} calls add up their effects.

Reimplemented from \texttt{Fl_Widget_Surface}.

12.34.3.6 \texttt{untranslate()}

\begin{verbatim}
void Fl_EPS_File_Surface::untranslate (  
    void ) \text{ [virtual]}
\end{verbatim}

Undoes the effect of a previous \texttt{translate()} call.

Reimplemented from \texttt{Fl_Widget_Surface}.

The documentation for this class was generated from the following file:

\begin{itemize}
\item \texttt{Fl_PostScript.H}
\end{itemize}

12.35 \texttt{Fl_File_Browser} Class Reference

The \texttt{Fl_File_Browser} widget displays a list of filenames, optionally with file-specific icons.

\begin{verbatim}
#include <Fl_File_Browser.H>
\end{verbatim}

Inheritance diagram for \texttt{Fl_File_Browser}:
Public Types

• enum { FILES, DIRECTORIES }

Public Types inherited from Fl_Browser

• enum Fl_Line_Position { TOP, BOTTOM, MIDDLE }
  For internal use only?

Public Types inherited from Fl_Browser_

• enum {
  HORIZONTAL = 1, VERTICAL = 2, BOTH = 3, ALWAYS_ON = 4,
  HORIZONTAL_ALWAYS = 5, VERTICAL_ALWAYS = 6, BOTH_ALWAYS = 7 }
  Values for has_scrollbar().

Public Member Functions

• const char ∗ errmsg () const
  Returns OS error messages, or NULL if none.
• void errmsg (const char ∗ emsg)
  Sets OS error message to a string, which can be NULL.
• int filetype () const
  Sets or gets the file browser type, FILES or DIRECTORIES.
• void filetype (int t)
  Sets or gets the file browser type, FILES or DIRECTORIES.
• const char ∗ filter () const
  Sets or gets the filename filter.
• void filter (const char ∗ pattern)
  Sets or gets the filename filter.
• Fl_File_Browser (int, int, int, int, const char ∗=0)
  The constructor creates the Fl_File_Browser widget at the specified position and size.
• uchar iconsize () const
  Sets or gets the size of the icons.
• void iconsize (uchar s)
  Sets or gets the size of the icons.
• int load (const char ∗ directory, Fl_File_Sort_F ∗ sort=fl_numericsort)
  Loads the specified directory into the browser.
• Fl_Fontsize textsize () const
• void textsize (Fl_Fontsize s)
Public Member Functions inherited from Fl_Browser

- void add (const char *newtext, void *d=0)
  Adds a new line to the end of the browser.
- void bottomline (int line)
  Scrolls the browser so the bottom item in the browser is showing the specified line.
- void clear ()
  Removes all the lines in the browser.
- char column_char () const
  Gets the current column separator character.
- void column_char (char c)
  Sets the column separator to c.
- const int *column_widths () const
  Gets the current column width array.
- void column_widths (const int *arr)
  Sets the current array to arr.
- void *data (int line) const
  Returns the user data() for specified line.
- void data (int line, void *d)
  Sets the user data for specified line to d.
- void display (int line, int val=1)
  For back compatibility.
- int displayed (int line) const
  Returns non-zero if line has been scrolled to a position where it is being displayed.
- Fl_Browser (int X, int Y, int W, int H, const char *L=0)
  The constructor makes an empty browser.
- char format_char () const
  Gets the current format code prefix character, which by default is '@'.
- void format_char (char c)
  Sets the current format code prefix character to c.
- void hide () FL_OVERRIDE
  Hides the entire Fl_Browser widget – opposite of show().
- void hide (int line)
  Makes line invisible, preventing selection by the user.
- Fl_Image *icon (int line) const
  Returns the icon currently defined for line.
- void icon (int line, Fl_Image *icon)
  Set the image icon for line to the value icon.
- void insert (int line, const char *newtext, void *d=0)
  Insert a new entry whose label is newtext above given line, optional data d.
- void lineposition (int line, Fl_Line_Position pos)
  Updates the browser so that line is shown at position pos.
- int load (const char *filename)
  Clears the browser and reads the file, adding each line from the file to the browser.
- void make_visible (int line)
  Make the item at the specified line visible().
- void middleline (int line)
  Scrolls the browser so the middle item in the browser is showing the specified line.
- void move (int to, int from)
  Line from is removed and reinserted at to.
- void remove (int line)
Remove entry for given line number, making the browser one line shorter.

- **void remove_icon (int line)**
  Removes the icon for line.

- **void replace (int a, const char ∗b)**
  For back compatibility only.

- **int select (int line, int val=1)**
  Sets the selection state of the item at line to the value val.

- **int selected (int line) const**
  Returns 1 if specified line is selected, 0 if not.

- **void show () FL_OVERRIDE**
  Shows the entire Fl_Browser widget – opposite of hide().

- **void show (int line)**
  Makes line visible, and available for selection by user.

- **int size () const**
  Returns how many lines are in the browser.

- **void size (int W, int H)**

- **void swap (int a, int b)**
  Swaps two browser lines a and b.

- **const char ∗text (int line) const**
  Returns the label text for the specified line.

- **void text (int line, const char ∗newtext)**
  Sets the text for the specified line to newtext.

- **Fl_Fontsize textsize () const**
  Gets the default text size (in pixels) for the lines in the browser.

- **void textsize (Fl_Fontsize newSize)**
  Sets the default text size (in pixels) for the lines in the browser to newSize.

- **int topline () const**
  Returns the line that is currently visible at the top of the browser.

- **void topline (int line)**
  Scrolls the browser so the top item in the browser is showing the specified line.

- **int value () const**
  Returns the line number of the currently selected line, or 0 if none selected.

- **void value (int line)**
  Sets the browser’s value(), which selects the specified line.

- **int visible (int line) const**
  Returns non-zero if the specified line is visible, 0 if hidden.

- **~Fl_Browser ()**
  The destructor deletes all list items and destroys the browser.

**Public Member Functions inherited from Fl_Browser_**

- **int deselect (int docallbacks=0)**
  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.

- **void display (void ∗item)**
  Displays the item, scrolling the list as necessary.

- **int handle (int event) FL_OVERRIDE**
  Handles the event within the normal widget bounding box.

- **uchar has_scrollbar () const**
  Returns the current scrollbar mode, see Fl_Browser_::has_scrollbar(uchar)

- **void has_scrollbar (uchar mode)**
  Sets whether the widget should have scrollbars or not (default Fl_Browser_::BOTH).
• int hposition () const
  Gets the horizontal scroll position of the list as a pixel position pos.

• void hposition (int)
  Sets the horizontal scroll position of the list to pixel position pos.

• int linespacing () const
  Return the height of additional spacing between browser lines.

• void linespacing (int pixels)
  Add some space between browser lines.

• int position () const
• void position (int pos)
  • void position (int x, int y)

• void resize (int X, int Y, int W, int H) FL_OVERRIDE
  Repositions and/or resizes the browser.

• void scrollbar_left ()
  Moves the vertical scrollbar to the lefthand side of the list.

• void scrollbar_right ()
  Moves the vertical scrollbar to the righthand side of the list.

• int scrollbar_size () const
  Gets the current size of the scrollbars’ troughs, in pixels.

• void scrollbar_size (int newSize)
  Sets the pixel size of the scrollbars’ troughs to newSize, in pixels.

• int scrollbar_width () const
  Returns the global value Fl::scrollbar_size().

• void scrollbar_width (int width)
  Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

• int select (void ∗item, int val=1, int docallbacks=0)
  Sets the selection state of item to val, and returns 1 if the state changed or 0 if it did not.

• int select_only (void ∗item, int docallbacks=0)
  Selects item and returns 1 if the state changed or 0 if it did not.

• void sort (int flags=0)
  Sort the items in the browser based on flags.

• Fl_Color textcolor () const
  Gets the default text color for the lines in the browser.

• void textcolor (Fl_Color col)
  Sets the default text color for the lines in the browser to color col.

• Fl_Font textfont () const
  Gets the default text font for the lines in the browser.

• void textfont (Fl_Font font)
  Sets the default text font for the lines in the browser to font.

• Fl_Fontsize fontsize () const
  Gets the default text size (in pixels) for the lines in the browser.

• void fontsize (Fl_Fontsize newSize)
  Sets the default text size (in pixels) for the lines in the browser to size.

• int vposition () const
  Gets the vertical scroll position of the list as a pixel position pos.

• void vposition (int pos)
  Sets the vertical scroll position of the list to pixel position pos.
Public Member Functions inherited from Fl_Group

- **Fl_Widget * _ddfdesign_kludge ()**
  
  This is for forms compatibility only.

- **void add (Fl_Widget *)**
  
  The widget is removed from its current group (if any) and then added to the end of this group.

- **void add (Fl_Widget *)**
  
  See void Fl_Group::add(Fl_Widget &w)

- **void add_resizable (Fl_Widget &)**
  
  Adds a widget to the group and makes it the resizable widget.

- **Fl_Widget * const * array () const**
  
  Returns a pointer to the array of children.

- **Fl_Group * as_group () const FL_OVERRIDE**
  
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- **void begin ()**
  
  Sets the current group so you can build the widget tree by just constructing the widgets.

- **Fl_Widget * child (int n) const**
  
  Returns array()[n].

- **int children () const**
  
  Returns how many child widgets the group has.

- **void clear ()**
  
  Deletes all child widgets from memory recursively.

- **unsigned int clip_children ()**
  
  Returns the current clipping mode.

- **void clip_children (int c)**
  
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- **virtual int delete_child (int n)**
  
  Removes the widget at index from the group and deletes it.

- **void end ()**
  
  Exactly the same as current(this->parent()).

- **int find (const Fl_Widget *) const**
  
  See int Fl_Group::find(const Fl_Widget *) const.

- **int find (const Fl_Widget *) const**
  
  Searches the child array for the widget and returns the index.

- **Fl_Group (int, int, int, int, const char *)=0**
  
  Creates a new Fl_Group widget using the given position, size, and label string.

- **void focus (Fl_Widget *)**

- **void forms_end ()**
  
  This is for forms compatibility only.

- **int handle (int) FL_OVERRIDE**
  
  Handles the specified event.

- **void init_sizes ()**
  
  Resets the internal array of widget sizes and positions.

- **void insert (Fl_Widget &, int i)**
  
  The widget is removed from its current group (if any) and then inserted into this group.

- **void insert (Fl_Widget &, Fl_Widget *)**
  
  This does insert(w, find(before)).

- **void remove (Fl_Widget &)**
  
  Removes a widget from the group but does not delete it.

- **void remove (Fl_Widget *)**
Removes the widget \( o \) from the group.

- void remove (int index)

  Removes the widget at \( index \) from the group but does not delete it.

- \*Fl_Widget resizable () const

  Returns the group's resizable widget.

- void resizable (\*Fl_Widget &o)

  Sets the group's resizable widget.

- void resizable (\*Fl_Widget o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- void resize (int, int, int, int) FL_OVERRIDE

  Resizes the Fl_Group widget and all of its children.

- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()

- void _set_fullscreen ()

- void activate ()

  Activates the widget.

- unsigned int active () const

  Returns whether the widget is active.

- int active_r () const

  Returns whether the widget and all of its parents are active.

- Fl_Align align () const

  Gets the label alignment.

- void align (Fl_Align alignment)

  Sets the label alignment.

- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.

- virtual class Fl_Gl_Window *as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const *as_gl_window () const

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual Fl_Window *as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const *as_window () const

- void bind_deimage (Fl_Img *img)

  Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image (Fl_Img *img)

  Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- Fl_Boxtype box () const

  Gets the box type of the widget.

- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.

- Fl_Callback_p callback () const
Gets the current callback function for the widget.
• void callback (Fl_Callback ∗cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback ∗cb, void ∗p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 ∗cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 ∗cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
  Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget ∗w) const
  Checks if w is a child of this widget.
• void copy_label (const char ∗new_label)
  Sets the current label.
• void copy_tooltip (const char ∗text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• **Fl_Image * deimage ()**
  Gets the image that is used as part of the widget label when in the inactive state.

• **const Fl_Image * deimage () const**
  Gets the image that is used as part of the widget label when in the inactive state.

• **void deimage (Fl_Image &img)**
  Sets the image to use as part of the widget label when in the inactive state.

• **void deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the inactive state.

• **int deimage_bound () const**
  Returns whether the inactive image is managed by the widget.

• **void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with default arguments.

• **void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

• **void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

• **void draw_label (int, int, int, int, Fl_Align) const**
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• **int h () const**
  Gets the widget height.

• **Fl_Image * image ()**
  Gets the image that is used as part of the widget label when in the active state.

• **const Fl_Image * image () const**
  Gets the image that is used as part of the widget label when in the active state.

• **void image (Fl_Image &img)**
  Sets the image to use as part of the widget label when in the active state.

• **void image (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the active state.

• **int image_bound () const**
  Returns whether the image is managed by the widget.

• **int inside (const Fl_Widget *wgt) const**
  Checks if this widget is a child of wgt.

• **int is_label_copied () const**
  Returns whether the current label was assigned with copy_label().

• **const char * label () const**
  Gets the current label text.

• **void label (const char *text)**
  Sets the current label pointer.

• **void label (Fl_Labeltype a, const char *b)**
  Shortcut to set the label text and type in one call.

• **Fl_Color labelcolor () const**
  Gets the label color.

• **void labelcolor (Fl_Color c)**
  Sets the label color.

• **Fl_Font labelfont () const**
  Gets the font to use.

• **void labelfont (Fl_Font f)**
  Sets the font to use.

• **Fl_Fontsize labelsize () const**
  Gets the font size in pixels.

• **void labelsize (Fl_Fontsize pix)**
Sets the font size in pixels.

- **Fl_Labeltype labeltype () const**
  
  Gets the label type.

- **void labeltype (Fl_Labeltype a)**
  
  Sets the label type.

- **void measure_label (int &ww, int &hh) const**
  
  Sets width \( \text{ww} \) and height \( \text{hh} \) accordingly with the label size.

- **bool needs_keyboard () const**
  
  Returns whether this widget needs a keyboard.

- **void needs_keyboard (bool needs)**
  
  Sets whether this widget needs a keyboard.

- **unsigned int output () const**
  
  Returns if a widget is used for output only.

- **Fl_Group * parent () const**
  
  Returns a pointer to the parent widget.

- **void parent (Fl_Group *p)**
  
  Internal use only - “for hacks only”.

- **void position (int X, int Y)**
  
  Repositions the window or widget.

- **void redraw ()**
  
  Schedules the drawing of the widget.

- **void redraw_label ()**
  
  Schedules the drawing of the label.

- **Fl_Color selection_color () const**
  
  Gets the selection color.

- **void selection_color (Fl_Color a)**
  
  Sets the selection color.

- **void set_active ()**
  
  Marks the widget as active without sending events or changing focus.

- **void set_changed ()**
  
  Marks the value of the widget as changed.

- **void set_output ()**
  
  Sets a widget to output only.

- **void set_visible ()**
  
  Makes the widget visible.

- **void set_visible_focus ()**
  
  Enables keyboard focus navigation with this widget.

- **int shortcut_label () const**
  
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.

- **void shortcut_label (int value)**
  
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.

- **void size (int W, int H)**
  
  Changes the size of the widget.

- **int take_focus ()**
  
  Gives the widget the keyboard focus.

- **unsigned int takesevents () const**
  
  Returns if the widget is able to take events.

- **int test_shortcut ()**
  
  Returns true if the widget’s label contains the entered ‘&x’ shortcut.

- **const char * tooltip () const**
  
  Gets the current tooltip text.
• void tooltip (const char *text)
  Sets the current tooltip text.

• Fl_Window *top_window () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window *top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const
  Gets the widget type.

• void type (uchar t)
  Sets the widget type.

• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void *user_data () const
  Gets the user data for this widget.

• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

• void user_data (void *v)
  Sets the user data for this widget.

• unsigned int visible () const
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

• void visible_focus (int v)
  Modifies keyboard focus navigation.

• int visible_r () const
  Returns whether a widget and all its parents are visible.

• int w () const
  Gets the widget width.

• Fl_When when () const
  Returns the conditions under which the callback is called.

• void when (uchar i)
  Sets the flags used to decide when a callback is called.

• Fl_Window *window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  Gets the widget position in its window.

• int y () const
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

• static Fl_Group *current ()
  Returns the currently active group.

• static void current (Fl_Group *g)
  Sets the current group.
Static Public Member Functions inherited from `Fl_Widget`

- static void `default_callback` (Fl_Widget ∗widget, void ∗data)
  
  The default callback for all widgets that don't set a callback.
- static unsigned int `label_shortcut` (const char ∗t)
  
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int `test_shortcut` (const char ∗t, const bool require_alt=false)
  
  Returns true if the given text t contains the entered '&x' shortcut.

Public Attributes inherited from `Fl_Browser`

- Fl_Scrollbar `hscrollbar`
  
  Horizontal scrollbar.
- Fl_Scrollbar `scrollbar`
  
  Vertical scrollbar.

Protected Types inherited from `Fl_Widget`

```cpp
enum {
    INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
    FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
    OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
    MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
    GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
    NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
    AUTO_DELETE_USER_DATA = 1<<23 ,
    MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
```

Protected Member Functions inherited from `Fl_Browser`

- FL_BLINE ∗_remove (int line)
  
  Removes the item at the specified line.
- FL_BLINE ∗find_line (int line) const
  
  Returns the item for specified line.
- void insert (int line, FL_BLINE ∗item)
  
  Insert specified item above line.
- void ∗item_at (int line) const FL_OVERRIDE
  
  Return the item at specified line.
- void ∗item_first () const FL_OVERRIDE
  
  Return the very first item in the list.
- void ∗item_last () const FL_OVERRIDE
  
  Returns the very last item in the list.
- void ∗item_next (void ∗item) const FL_OVERRIDE
  
  Returns the next item after item.
- void ∗item_prev (void ∗item) const FL_OVERRIDE
  
  Returns the previous item before item.
- void item_select (void ∗item, int val) FL_OVERRIDE
  
  Change the selection state of item to the value val.
- int item_selected (void ∗item) const FL_OVERRIDE
See if item is selected.

- void item_swap (void ∗a, void ∗b) FL_OVERRIDE
  Swap the items a and b.
- const char ∗ item_text (void ∗item) const FL_OVERRIDE
  Returns the label text for item.
- int lineno (void ∗item) const
  Returns line number corresponding to item, or zero if not found.
- void swap (FL_BLINE ∗a, FL_BLINE ∗b)
  Swap the two items a and b.

Protected Member Functions inherited from Fl_Browser_

- void bbox (int &X, int &Y, int &W, int &H) const
  Returns the bounding box for the interior of the list's display window, inside the scrollbars.
- void deleting (void ∗item)
  This method should be used when item is being deleted from the list.
- int displayed (void ∗item) const
  Returns non-zero if item has been scrolled to a position where it is being displayed.
- void draw () FL_OVERRIDE
  Draws the list within the normal widget bounding box.
- void ∗find_item (int ypos)
  This method returns the item under mouse y position ypos.
- Fl_Browser_ (int X, int Y, int W, int H, const char ∗L=0)
  The constructor makes an empty browser.
- virtual int full_width () const
  This method may be provided by the subclass to indicate the full width of the item list, in pixels.
- void inserting (void ∗a, void ∗b)
  This method should be used when an item is in the process of being inserted into the list.
- virtual int item_quick_height (void ∗item) const
  This method may be provided by the subclass to return the height of the item, in pixels.
- int leftedge () const
  This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.
- void new_list ()
  This method should be called when the list data is completely replaced or cleared.
- void redraw_line (void ∗item)
  This method should be called when the contents of item has changed, but not its height.
- void redraw_lines ()
  This method will cause the entire list to be redrawn.
- void replacing (void ∗a, void ∗b)
  This method should be used when item a is being replaced by item b.
- void ∗selection () const
  Returns the item currently selected, or NULL if there is no selection.
- void swapping (void ∗a, void ∗b)
  This method should be used when two items a and b are being swapped.
- void ∗top () const
  Returns the item that appears at the top of the list.
Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
- void draw () FL_OVERRIDE
  Draws the widget.
- void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.
- void draw_children ()
  Draws all children of the group.
- void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)
  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()
  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void draw_label () const
  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.
- unsigned int flags () const
  Gets the widget flags mask.
- void h (int v)
  Internal use only.
- void set_flag (unsigned int c)
Sets a flag in the flags mask.

- void \(w\) (int \(v\))
  
  *Internal use only.*

- void \(x\) (int \(v\))
  
  *Internal use only.*

- void \(y\) (int \(v\))
  
  *Internal use only.*

### 12.35.1 Detailed Description

The `Fl_File_Browser` widget displays a list of filenames, optionally with file-specific icons.

### 12.35.2 Constructor & Destructor Documentation

#### 12.35.2.1 Fl_File_Browser()

```cpp
Fl_File_Browser::Fl_File_Browser (
    int \(X\),
    int \(Y\),
    int \(W\),
    int \(H\),
    const char * \(l = 0\))
```

The constructor creates the `Fl_File_Browser` widget at the specified position and size. The destructor destroys the widget and frees all memory that has been allocated.

### 12.35.3 Member Function Documentation

#### 12.35.3.1 errmsg() [1/2]

```cpp
const char * Fl_File_Browser::errmsg ( ) const [inline]
```

Returns OS error messages, or NULL if none. Use when advised.

#### 12.35.3.2 errmsg() [2/2]

```cpp
void Fl_File_Browser::errmsg (const char * \(emsg\))
```

Sets OS error message to a string, which can be NULL. Frees previous if any. void `errmsg(const char *emsg);`

#### 12.35.3.3 filetype() [1/2]

```cpp
int Fl_File_Browser::filetype ( ) const [inline]
```

Sets or gets the file browser type, FILES or DIRECTORIES. When set to FILES, both files and directories are shown. Otherwise only directories are shown.

#### 12.35.3.4 filetype() [2/2]

```cpp
void Fl_File_Browser::filetype (int \(t\)) [inline]
```

Sets or gets the file browser type, FILES or DIRECTORIES. When set to FILES, both files and directories are shown. Otherwise only directories are shown.

#### 12.35.3.5 filter() [1/2]

```cpp
const char * Fl_File_Browser::filter ( ) const [inline]
```

Sets or gets the filename filter. The pattern matching uses the `fl_filename_match()` function in FLTK.
12.35.3.6 filter() [2/2]
void Fl_File_Browser::filter (const char * pattern)
Sets or gets the filename filter.
The pattern matching uses the fl_filename_match() function in FLTK.

12.35.3.7 iconsize() [1/2]
uchar Fl_File_Browser::iconsize ( ) const [inline]
Sets or gets the size of the icons.
The default size is 20 pixels.

12.35.3.8 iconsize() [2/2]
void Fl_File_Browser::iconsize (uchar s) [inline]
Sets or gets the size of the icons.
The default size is 20 pixels.

12.35.3.9 load()
int Fl_File_Browser::load (const char * directory, Fl_File_Sort_F * sort = fl_numericsort)
Loads the specified directory into the browser.
If icons have been loaded then the correct icon is associated with each file in the list.
If directory is "", all mount points (unix) or drive letters (Windows) are listed.
The sort argument specifies a sort function to be used with fl_filename_list().
Return value is the number of filename entries, or 0 if none. On error, 0 is returned, and errmsg() has OS error string if non-NULL.
The documentation for this class was generated from the following files:
  • Fl_File_Browser.H
  • Fl_File_Browser.cxx

12.36 Fl_File_Chooser Class Reference
The Fl_File_Chooser widget displays a standard file selection dialog that supports various selection modes.

Public Types
  • enum Type { SINGLE = 0 , MULTI = 1 , CREATE = 2 , DIRECTORY = 4 }
    Determines the type of file chooser presented to the user.

Public Member Functions
  • Fl_Widget * add_extra (Fl_Widget *gr)
    Adds an extra widget at the bottom of the Fl_File_Chooser window.
  • void callback (void (*)(Fl_File_Chooser *, void *), void *d=0)
    Sets the file chooser callback cb and associated data d.
  • Fl_Color color ()
    Gets the background color of the Fl_File_Browser list.
  • void color (Fl_Color c)
    Sets the background color of the Fl_File_Browser list.
  • int count ()
Returns the number of selected files.

- char * directory ()
  Gets the current directory.
- void directory (const char *d)
  Sets the current directory.
- const char * filter ()
  Gets the current filename filter patterns.
- void filter (const char *p)
  Sets the current filename filter patterns.
- int filter_value ()
  Gets the current filename filter selection.
- void filter_value (int f)
  Sets the current filename filter selection.

**Fl_File_Chooser** (const char *pathname, const char *pattern, int type_val, const char *title)

The constructor creates the Fl_File_Chooser dialog shown.

- int h () const
  Hides the Fl_File_Chooser window.
- void hide ()

- uchar iconsize ()
  Gets the size of the icons in the Fl_File_Browser.
- void iconsize (uchar s)
  Sets the size of the icons in the Fl_File_Browser.
- const char * label ()
  Gets the title bar text for the Fl_File_Chooser.
- void label (const char *l)
  Sets the title bar text for the Fl_File_Chooser.
- const char * ok_label ()
  Gets the label for the "ok" button in the Fl_File_Chooser.
- void ok_label (const char *l)
  Sets the label for the "ok" button in the Fl_File_Chooser.

- int preview () const
  Returns the current state of the preview box.
- void preview (int e)
  Enable or disable the preview tile.

- void rescan ()
  Reloads the current directory in the Fl_File_Browser.
- void rescan_keep_filename ()
  Rescan the current directory without clearing the filename, then select the file if it is in the list.

- void resize (int x, int y, int w, int h)
- void show ()
  Shows the Fl_File_Chooser window.

- int shown ()
  Returns non-zero if the file chooser main window show() has been called, but not hide().
- void size (int w, int h)

**Fl_Color textcolor ()**

- Gets the current Fl_File_Browser text color.
- void textcolor (Fl_Color c)
  Sets the current Fl_File_Browser text color.

**Fl_Font textfont ()**

- Gets the current Fl_File_Browser text font.
• void **textfont** (Fl_Font f)
  Sets the current Fl_File_Browser text font.

• Fl_Fontsize **textsize** ()
  Gets the current Fl_File_Browser text size.

• void **textsize** (Fl_Fontsize s)
  Sets the current Fl_File_Browser text size.

• int **type** ()
  Gets the current type of Fl_File_Chooser.

• void **type** (int t)
  Sets the current type of Fl_File_Chooser.

• void * **user_data** () const
  Gets the file chooser user data.

• void **user_data** (void * d)
  Sets the file chooser user data d.

• void **value** (const char * filename)
  Sets the current value of the selected file.

• const char * **value** (int f=1)
  Gets the current value of the selected file(s).

• int **visible** ()
  Returns 1 if the Fl_File_Chooser window is visible.

• int w () const
• int x () const
• int y () const

• ~Fl_File_Chooser ()
  Destroys the widget and frees all memory used by it.

**Public Attributes**

• Fl_Button * newButton
  The "new directory" button is exported so that application developers can control the appearance and use.

• Fl_Check_Button * previewButton
  The "preview" button is exported so that application developers can control the appearance and use.

• Fl_Check_Button * showHiddenButton
  When checked, hidden files (i.e., filename begins with dot) are displayed.

**Static Public Attributes**

• static const char * add_favorites_label = "Add to Favorites"
  [standard text may be customized at run-time]

• static const char * all_files_label = "All Files (+)"
  [standard text may be customized at run-time]

• static const char * custom_filter_label = "Custom Filter"
  [standard text may be customized at run-time]

• static const char * existing_file_label = "Please choose an existing file!"
  [standard text may be customized at run-time]

• static const char * favorites_label = "Favorites"
  [standard text may be customized at run-time]

• static const char * filename_label = "Filename:"
  [standard text may be customized at run-time]

• static const char * filesystems_label = Fl::system_driver()->filesystems_label()
  [standard text may be customized at run-time]

• static const char * hidden_label = "Show hidden files"
• static const char * manage_favorites_label = "Manage Favorites"
• static const char * new_directory_label = "New Directory?"
• static const char * new_directory_tooltip = "Create a new directory."
• static const char * preview_label = "Preview"
• static const char * save_label = "Save"
• static Fl_File_Sort_F * sort = fl_numericsort
  
  the sort function that is used when loading the contents of a directory.

Protected Member Functions
• void show_error_box (int val)
  Show error box if val=1, hide if val=0.

Related Symbols
(Note that these are not member symbols.)
• char * fl_dir_chooser (const char *message, const char *fname, int relative)
  Shows a file chooser dialog and gets a directory.
• char * fl_file_chooser (const char *message, const char *pat, const char *fname, int relative)
  Shows a file chooser dialog and gets a filename.
• void fl_file_chooser_callback (void(*cb)(const char *))
  Set the file chooser callback.
• void fl_file_chooser_ok_label (const char *l)
  Set the "OK" button label.

12.36.1 Detailed Description

The Fl_File_Chooser widget displays a standard file selection dialog that supports various selection modes.
Features include:

- Multiple filter patterns can be specified, with parenthesis around filters, and tabs to separate each pattern, e.g.:
  ```
  char pattern[] = "Image Files (*.bmp,gif,jpg,png,xbm,xpm)\t"
  "Web Files (*.htm,html,php)\t"
  "All Files (*)\t"
  ``
- If no "*" pattern is provided, then an entry for "All Files (*)" is automatically added.
- An optional file preview box is provided which can be toggled by programmer or user showing images, or the first 2048 bytes of printable text.
- Preview image loading functions can be registered to provide custom file previews.
- The favorites button shows up to 100 user-saved favorite directories, the user's home directory, and a filesystems item.
- A simple dialog is provided for managing saved directories.
- Shortcut keys are provided:

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+a</td>
<td>Adds a directory to the favorites list</td>
</tr>
<tr>
<td>Alt+m</td>
<td>Manages the favorites list</td>
</tr>
<tr>
<td>Alt+f</td>
<td>Shows the filesystem list</td>
</tr>
<tr>
<td>Alt+h</td>
<td>Go to the home directory</td>
</tr>
<tr>
<td>Alt+0..9</td>
<td>going to any of the first 10 favorites</td>
</tr>
</tbody>
</table>

The `Fl_FileChooser` widget transmits UTF-8 encoded filenames to its user. It is recommended to open files that may have non-ASCII names with the `fl_fopen()` or `fl_open()` utility functions that handle these names in a cross-
platform way (whereas the standard fopen()/open() functions fail on the Windows platform to open files with a non-ASCII name).

The Fl_File_Chooser class also exports several static values that may be used to localize or customize the appearance of all file chooser dialogs:

<table>
<thead>
<tr>
<th>Member</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_favorites_label</td>
<td>&quot;Add to Favorites&quot;</td>
</tr>
<tr>
<td>all_files_label</td>
<td>&quot;All Files (+)&quot;</td>
</tr>
<tr>
<td>custom_filter_label</td>
<td>&quot;Custom Filter&quot;</td>
</tr>
<tr>
<td>existing_file_label</td>
<td>&quot;Please choose an existing file!&quot;</td>
</tr>
<tr>
<td>favorites_label</td>
<td>&quot;Favorites&quot;</td>
</tr>
<tr>
<td>filename_label</td>
<td>&quot;Filename:&quot;</td>
</tr>
<tr>
<td>filesystems_label</td>
<td>&quot;My Computer&quot; (Windows) &quot;File Systems&quot; (all others)</td>
</tr>
<tr>
<td>hidden_label</td>
<td>&quot;Show hidden files:&quot;</td>
</tr>
<tr>
<td>manage_favorites_label</td>
<td>&quot;Manage Favorites&quot;</td>
</tr>
<tr>
<td>new_directory_label</td>
<td>&quot;New Directory?&quot;</td>
</tr>
<tr>
<td>new_directory_tooltip</td>
<td>&quot;Create a new directory.&quot;</td>
</tr>
<tr>
<td>preview_label</td>
<td>&quot;Preview&quot;</td>
</tr>
<tr>
<td>save_label</td>
<td>&quot;Save&quot;</td>
</tr>
<tr>
<td>show_label</td>
<td>&quot;Show:&quot;</td>
</tr>
<tr>
<td>sort</td>
<td>fl_numericsort</td>
</tr>
</tbody>
</table>

The Fl_File_Chooser::sort member specifies the sort function that is used when loading the contents of a directory and can be customized at run-time.

The Fl_File_Chooser class also exports the Fl_File_Chooser::newButton and Fl_File_Chooser::previewButton widgets so that application developers can control their appearance and use.

### 12.36.2 Member Enumeration Documentation

#### 12.36.2.1 Type

```cpp
enum Fl_File_Chooser::Type
```

Determines the type of file chooser presented to the user.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINGLE</td>
<td>Select a single, existing file.</td>
</tr>
<tr>
<td>MULTI</td>
<td>Select one or more existing files.</td>
</tr>
<tr>
<td>CREATE</td>
<td>When used alone, select a single, existing file or specify a new filename. Can be combined with DIRECTORY (e.g. CREATE/DIRECTORY) to have the same effect for directories.</td>
</tr>
<tr>
<td>DIRECTORY</td>
<td>Select a single, existing directory. Can be combined with CREATE.</td>
</tr>
</tbody>
</table>

### 12.36.3 Constructor & Destructor Documentation

#### 12.36.3.1 Fl_File_Chooser()

`Fl_File_Chooser::Fl_File_Chooser ( const char *pathname, const char *pattern, int type_val, const char *title )`
The constructor creates the Fl_File_Chooser dialog shown.

- The **pathname** argument can be a directory name or a complete file name (in which case the corresponding file is highlighted in the list and in the filename input field.)

- The **pattern** argument can be a NULL string or "*" to list all files, or it can be a series of descriptions and filter strings separated by tab characters (\t). The format of filters is either "Description text (patterns)" or just "patterns". A file chooser that provides filters for HTML and image files might look like:
  
  "HTML Files (*.html)\tImage Files (*.bmp,gif,jpg,png)"

  The file chooser will automatically add the "All Files (*)" pattern to the end of the string you pass if you do not provide one. The first filter in the string is the default filter. See the FLTK documentation on `fl_filename_match()` for the kinds of pattern strings that are supported.

- The **type_val** argument can be one of the `Fl_File_Chooser::Type` values.

- The **title** argument is used to set the title bar text for the Fl_File_Chooser window.

### 12.36.4 Member Function Documentation

#### 12.36.4.1 `add_extra()`

```cpp
Fl_Widget * Fl_File_Chooser::add_extra (Fl_Widget * extra)
```

Adds an extra widget at the bottom of the Fl_File_Chooser window. You can use any `Fl_Widget` or `Fl_Group`. If you use an `Fl_Group`, set its (x, y) coordinates to (0, 0) and position its children relative to (0, 0) inside the `Fl_Group` container widget. Make sure that all child widgets of the `Fl_Group` are entirely included inside the bounding box of their parents, i.e. the `Fl_Group` widget, and the Fl_File_Chooser window, respectively.

**Note**

The width of the Fl_File_Chooser window is an undocumented implementation detail and may change in the future.

If `extra` is NULL any previous extra widget is removed.

**Parameters**

```
in extra Custom widget or group to be added to the Fl_File_Chooser window.
```

**Returns**

Pointer to previous extra widget or NULL if not set previously.

**Note**

`Fl_File_Chooser` does **not** delete the extra widget in its destructor! The extra widget is removed from the Fl_File_Chooser window before the Fl_File_Chooser widget gets destroyed. To prevent memory leakage, don't forget to delete unused extra widgets.

#### 12.36.4.2 `filter()`

```cpp
void Fl_File_Chooser::filter (const char * p)
```

Sets the current filename filter patterns. The filter patterns use `fl_filename_match()`. Multiple patterns can be used by separating them with tabs, like "*.jpg\t*.png\t*.gif\t*". In addition, you can provide human-readable labels with the patterns inside parenthesis, like "JPEG Files (*.jpg)\tPNG Files (*.png)\tGIF Files (*.gif)\tAll Files (*)". Use filter(NULL) to show all files.
12.36.4.3 iconsize() [1/2]

uchar Fl_File_Chooser::iconsize ( )

Gets the size of the icons in the Fl_File_Browser.
By default the icon size is set to 1.5 times the textsize().

12.36.4.4 iconsize() [2/2]

void Fl_File_Chooser::iconsize ( uchar s )

Sets the size of the icons in the Fl_File_Browser.
By default the icon size is set to 1.5 times the textsize().

12.36.4.5 preview()

void Fl_File_Chooser::preview ( int e )

Enable or disable the preview tile.
1 = enable preview, 0 = disable preview.

12.36.4.6 shown()

int Fl_File_Chooser::shown ( )

Returns non-zero if the file chooser main window show() has been called, but not hide().

See also

Fl_Window::shown()

12.36.4.7 value()

const char * Fl_File_Chooser::value ( int f = 1 )

Gets the current value of the selected file(s).
f is a 1-based index into a list of file names. The number of selected files is returned by Fl_File_Chooser::count().
This sample code loops through all selected files:

// Get list of filenames user selected from a MULTI chooser
for ( int t=1; t<chooser->count(); t++ ) {
    const char *filename = chooser->value(t);
    ...
}

12.36.5 Member Data Documentation

12.36.5.1 showHiddenButton

Fl_File_Chooser::showHiddenButton

When checked, hidden files (i.e., filename begins with dot) are displayed.
The "showHiddenButton" button is exported so that application developers can control its appearance.
The documentation for this class was generated from the following files:

• Fl_File_Chooser.H
• Fl_File_Chooser.cxx
• Fl_File_Chooser2.cxx
• fl_file_dir.cxx

12.37 Fl_File_Icon Class Reference

The Fl_File_Icon class manages icon images that can be used as labels in other widgets and as icons in the
FileBrowser widget.
#include <Fl_File_Icon.H>
Public Types

- enum {
  ANY, PLAIN, FIFO, DEVICE,
  LINK, DIRECTORY
}
- enum {
  END, COLOR, LINE, CLOSEDLINE,
  POLYGON, OUTLINEPOLYGON, VERTEX
}

Public Member Functions

- short ∗ add (short d)
  Adds a keyword value to the icon array, returning a pointer to it.
- short ∗ add_color (Fl_Color c)
  Adds a color value to the icon array, returning a pointer to it.
- short ∗ add_vertex (float x, float y)
  Adds a vertex value to the icon array, returning a pointer to it.
- short ∗ add_vertex (int x, int y)
  Adds a vertex value to the icon array, returning a pointer to it.
- void clear ()
  Clears all icon data from the icon.
- void draw (int x, int y, int w, int h, Fl_Color ic, int active=1)
  Draws an icon in the indicated area.
- Fl_File_Icon (const char ∗ p, int t, int nd=0, short ∗ d=0)
  Creates a new Fl_File_Icon with the specified information.
- void label (Fl_Widget ∗ w)
  Applies the icon to the widget, registering the Fl_File_Icon label type as needed.
- void load (const char ∗ f)
  Loads the specified icon image.
- int load_fti (const char ∗ fti)
  Loads an SGI icon file.
- int load_image (const char ∗ i)
  Load an image icon file from an image filename.
- Fl_File_Icon ∗ next ()
  Returns next file icon object.
- const char ∗ pattern ()
  Returns the filename matching pattern for the icon.
- int size ()
  Returns the number of words of data used by the icon.
- int type ()
  Returns the filetype associated with the icon, which can be one of the following:
- short ∗ value ()
  Returns the data array for the icon.
- ∼Fl_File_Icon ()
  The destructor destroys the icon and frees all memory that has been allocated for it.

Static Public Member Functions

- static Fl_File_Icon ∗ find (const char ∗ filename, int filetype=ANY)
  Finds an icon that matches the given filename and file type.
- static Fl_File_Icon ∗ first ()
  Returns a pointer to the first icon in the list.
- static void labeltype (const Fl_Label ∗ o, int x, int y, int w, int h, Fl_Align a)
Draw the icon label.

- static void load_system_icons (void)
  Loads all system-defined icons.

### 12.37.1 Detailed Description

The Fl_File_Icon class manages icon images that can be used as labels in other widgets and as icons in the FileBrowser widget.

### 12.37.2 Constructor & Destructor Documentation

#### 12.37.2.1 Fl_File_Icon()

```cpp
Fl_File_Icon::Fl_File_Icon (const char * p, int t, int nd = 0, short * d = 0)
```

Creates a new Fl_File_Icon with the specified information.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>p</th>
<th>filename pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>t</td>
<td>file type</td>
</tr>
<tr>
<td>in</td>
<td>nd</td>
<td>number of data values</td>
</tr>
<tr>
<td>in</td>
<td>d</td>
<td>data values</td>
</tr>
</tbody>
</table>

### 12.37.3 Member Function Documentation

#### 12.37.3.1 add()

```cpp
short * Fl_File_Icon::add (short d)
```

Adds a keyword value to the icon array, returning a pointer to it.

**Parameters**

| in | d | data value |

#### 12.37.3.2 add_color()

```cpp
short * Fl_File_Icon::add_color (Fl_Color c) [inline]
```

Adds a color value to the icon array, returning a pointer to it.

**Parameters**

| in | c | color value |

#### 12.37.3.3 add_vertex() [1/2]

```cpp
short * Fl_File_Icon::add_vertex (float x, float y) [inline]
```

Adds a vertex value to the icon array, returning a pointer to it.
The floating point version goes from 0.0 to 1.0. The origin (0.0) is in the lower-lefthand corner of the icon.

### Parameters

- **in x,y** vertex coordinates

#### 12.37.3.4 add_vertex() [2/2]

```c
short * Fl_File_Icon::add_vertex ( int x,
                                   int y ) [inline]
```

Adds a vertex value to the icon array, returning a pointer to it.

The integer version accepts coordinates from 0 to 10000. The origin (0.0) is in the lower-lefthand corner of the icon.

### Parameters

- **in x,y** vertex coordinates

#### 12.37.3.5 draw()

```c
void Fl_File_Icon::draw ( int x,
                         int y,
                         int w,
                         int h,
                         Fl_Color ic,
                         int active = 1 )
```

Draws an icon in the indicated area.

### Parameters

- **in x,y,w,h** position and size
- **in ic** icon color
- **in active** status, default is active [non-zero]

#### 12.37.3.6 find()

```c
Fl_File_Icon * Fl_File_Icon::find ( const char * filename,
                                    int filetype = ANY ) [static]
```

Finds an icon that matches the given filename and file type.

### Parameters

- **in filename** name of file
- **in filetype** enumerated file type

**Returns**

- matching file icon or NULL

#### 12.37.3.7 label()

```c
void Fl_File_Icon::label ( Fl_Widget * w )
```
Applies the icon to the widget, registering the Fl_File_Icon label type as needed.

Parameters

|   |   | widget for which this icon will become the label |

### 12.37.3.8 labeltype()

```cpp
def Fl_File_Icon::labeltype (  
    const Fl_Widget * o,  
    int x,  
    int y,  
    int w,  
    int h,  
    Fl_Align a ) [static]
```

Draw the icon label.

Parameters

|   |   | label data |
| in | o | label data |
| in | x,y,w,h | position and size of label |
| in | a | label alignment [not used] |

### 12.37.3.9 load()

```cpp
def Fl_File_Icon::load (  
    const char * f )
```

Loads the specified icon image. The format is deduced from the filename.

Parameters

|   |   | filename |

### 12.37.3.10 load_fti()

```cpp
def Fl_File_Icon::load_fti (  
    const char * fti )
```

Loads an SGI icon file.

Parameters

|   |   | icon filename |

Returns

0 on success, non-zero on error

### 12.37.3.11 load_image()

```cpp
def Fl_File_Icon::load_image (  
    const char * ifile )
```

Load an image icon file from an image filename.
Parameters

| in | ifile | image filename |

Returns

0 on success, non-zero on error

12.37.3.12 load_system_icons()

void Fl_File_Icon::load_system_icons (
    void ) [static]

Loads all system-defined icons.
This call is useful when using the FileChooser widget and should be used when the application starts:
Fl_File_Icon::load_system_icons();

12.37.3.13 next()

Fl_File_Icon * Fl_File_Icon::next ( ) [inline]
Returns next file icon object.
See Fl_File_Icon::first()

12.37.3.14 type()

int Fl_File_Icon::type ( ) [inline]
Returns the filetype associated with the icon, which can be one of the following:
- Fl_File_Icon::ANY, any kind of file.
- Fl_File_Icon::PLAIN, plain files.
- Fl_File_Icon::FIFO, named pipes.
- Fl_File_Icon::DEVICE, character and block devices.
- Fl_File_Icon::LINK, symbolic links.
- Fl_File_Icon::DIRECTORY, directories.

The documentation for this class was generated from the following files:
- Fl_File_Ic on.H
- Fl_File_Icon.cxx
- Fl_File_Icon2.cxx

12.38 Fl_File_Input Class Reference

This widget displays a pathname in a text input field.
#include <Fl_File_Input.H>
Inheritance diagram for Fl_File_Input:
Public Member Functions

- **Fl_Boxtype** `down_box ()` const
  
  Gets the box type used for the navigation bar.

- **void** `down_box (Fl_Boxtype b)`
  
  Sets the box type to use for the navigation bar.

- **Fl_Color** `errorcolor ()` const
  
  Gets the current error color.

- **void** `errorcolor (Fl_Color c)`
  
  Sets the current error color to `c`.

- **Fl_File_Input** (int X, int Y, int W, int H, const char ∗L=0)
  
  Creates a new Fl_File_Input widget using the given position, size, and label string.

- **int** `handle (int event) FL_OVERRIDE`
  
  Handle events in the widget.

- **const char ∗** `value ()`
  
  Returns the current value, which is a pointer to an internal buffer and is valid only until the next event is handled.

- **int** `value (const char ∗str)`
  
  Sets the value of the widget given a new string value.

- **int** `value (const char ∗str, int len)`
  
  Sets the value of the widget given a new string value and its length.

Public Member Functions inherited from **Fl_Input**

- **Fl_Input** (int, int, int, int, const char ∗=0)
  
  Creates a new Fl_Input widget using the given position, size, and label string.

- **int** `handle (int) FL_OVERRIDE`
  
  Handles the specified event.

Public Member Functions inherited from **Fl_Input_**

- **int** `append (const char ∗t, int l=0, char keep_selection=0)`
  
  Append text at the end.

- **bool** `can redo ()` const
  
  Check if there is a redo action available.

- **bool** `can undo ()` const
  
  Check if the last operation can be undone.

- **int** `copy (int clipboard)`
  
  Put the current selection into the clipboard.

- **int** `copy cuts ()`
  
  Copies the yank buffer to the clipboard.

- **Fl_Color** `cursor color ()` const
  
  Gets the color of the cursor.

- **void** `cursor color (Fl_Color n)`
  
  Sets the color of the cursor.

- **int** `cut ()`
  
  Deletes the current selection.

- **int** `cut (int a, int b)`
  
  Deletes all characters between index `a` and `b`.

- **int** `cut (int n)`
  
  Deletes the next `n` bytes rounded to characters before or after the cursor.

- **double** `dvalue ()` const
  
  Returns the widget text interpreted as a floating point number.
• Fl_Input_ (int, int, int, int, const char ∗=0)
  Creates a new Fl_Input_ widget.
• unsigned int index (int i) const
  Returns the character at index \( i \).
• int input_type () const
  Gets the input field type.
• void input_type (int t)
  Sets the input field type.
• int insert (const char ∗t, int l=0)
  Inserts text at the cursor position.
• int insert_position () const
  Gets the position of the text cursor.
• int insert_position (int p)
  Sets the cursor position and mark.
• int insert_position (int p, int m)
  Sets the index for the cursor and mark.
• int ivalue () const
  Returns the widget text interpreted as a signed integer.
• int mark () const
  Gets the current selection mark.
• int mark (int m)
  Sets the current selection mark.
• int maximum_size () const
  Gets the maximum length of the input field in characters.
• void maximum_size (int m)
  Sets the maximum length of the input field in characters.
• int position () const
• int position (int p)
• int position (int p, int m)
• int readonly () const
  Gets the read-only state of the input field.
• void readonly (int b)
  Sets the read-only state of the input field.
• int redo ()
  Redo previous undo operation.
• int replace (int b, int e, const char ∗text, int ilen=0)
  Deletes text from \( b \) to \( e \) and inserts the new string \( \text{text} \).
• void resize (int, int, int, int) FL_OVERRIDE
  Changes the size of the widget.
• int shortcut () const
  Return the shortcut key associated with this widget.
• void shortcut (int s)
  Sets the shortcut key associated with this widget.
• int size () const
  Returns the number of bytes in value().
• void size (int W, int H)
  Sets the width and height of this widget.
• int static_value (const char ∗)
  Changes the widget text.
• int static_value (const char ∗, int)
  Changes the widget text.
- int tab_nav () const  
  Gets whether the Tab key causes focus navigation in multiline input fields or not.
- void tab_nav (int val)  
  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- Fl_Color textcolor () const  
  Gets the color of the text in the input field.
- void textcolor (FL_Color n)  
  Sets the color of the text in the input field.
- Fl_Font textfont () const  
  Gets the font of the text in the input field.
- void textfont (FL_Font s)  
  Sets the font of the text in the input field.
- Fl_Fontsize textsize () const  
  Gets the size of the text in the input field.
- void textsize (FL_Fontsize s)  
  Sets the size of the text in the input field.
- int undo ()  
  Undoes previous changes to the text buffer.
- const char * value () const  
  Returns the text displayed in the widget.
- int value (const char *)  
  Changes the widget text.
- int value (const char *, int)  
  Changes the widget text.
- int value (double value)  
  Changes the widget text to a floating point number ("%g").
- int value (int value)  
  Changes the widget text to a signed integer number.
- int wrap () const  
  Gets the word wrapping state of the input field.
- void wrap (int b)  
  Sets the word wrapping state of the input field.
- ~Fl_Input_ ()  
  Destroys the widget.

Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()  
  Activates the widget.
- unsigned int active () const  
  Returns whether the widget is active.
- int active_r () const  
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const  
  Gets the label alignment.
- void align (FL_Align alignment)  
  Sets the label alignment.
- long argument () const  
  Gets the current user data (long) argument that is passed to the callback function.
• void argument (long v)
  *Sets the current user data (long) argument that is passed to the callback function.*

• virtual class Fl_Gl_Window ∗ as_gl_window ()
  *Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.*

• virtual class Fl_Gl_Window const ∗ as_gl_window () const

• virtual Fl_Group ∗ as_group ()
  *Returns an Fl_Group pointer if this widget is an Fl_Group.*

• virtual Fl_Group const ∗ as_group () const

• virtual Fl_Window ∗ as_window ()
  *Returns an Fl_Window pointer if this widget is an Fl_Window.*

• virtual Fl_Window const ∗ as_window () const

• void bind_deimage (Fl_Image ∗ img)
  *Sets the image to use as part of the widget label when in the inactive state.*

• void bind_deimage (int f)
  *Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.*

• void bind_image (Fl_Image ∗ img)
  *Sets the image to use as part of the widget label when in the active state.*

• void bind_image (int f)
  *Bind the image to the widget, so the widget will delete the image when it is no longer needed.*

• Fl_Boxtype box () const
  *Gets the box type of the widget.*

• void box (Fl_Boxtype new_box)
  *Sets the box type for the widget.*

• Fl_Callback_p callback () const
  *Gets the current callback function for the widget.*

• void callback (Fl_Callback ∗ cb)
  *Sets the current callback function for the widget.*

• void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  *Sets the current callback function and managed user data for the widget.*

• void callback (Fl_Callback ∗ cb, void ∗ p)
  *Sets the current callback function and data for the widget.*

• void callback (Fl_Callback0 ∗ cb)
  *Sets the current callback function for the widget.*

• void callback (Fl_Callback1 ∗ cb, long p=0)
  *Sets the current callback function for the widget.*

• unsigned int changed () const
  *Checks if the widget value changed since the last callback.*

• void clear_active ()
  *Marks the widget as inactive without sending events or changing focus.*

• void clear_changed ()
  *Marks the value of the widget as unchanged.*

• void clear_damage (uchar c=0)
  *Clears or sets the damage flags.*

• void clear_output ()
  *Sets a widget to accept input.*

• void clear_visible ()
  *Hides the widget.*

• void clear_visible_focus ()
  *Disables keyboard focus navigation with this widget.*

• Fl_Color color () const
  *Gets the background color of the widget.*
• void color (Fl_Color bg)
  Sets the background color of the widget.

• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

• Fl_Color color2 () const
  For back compatibility only.

• void color2 (unsigned a)
  For back compatibility only.

• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.

• void copy_label (const char *new_label)
  Sets the current label.

• void copy_tooltip (const char *text)
  Sets the current tooltip text.

• uchar damage () const
  Returns non-zero if draw() needs to be called.

• void damage (uchar c)
  Sets the damage bits for the widget.

• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

• int damage_resize (int, int, int, int)
  Internal use only.

• void deactivate ()
  Deactivates the widget.

• Fl_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

• const Fl_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• int h () const
  Gets the widget height.

• virtual void hide ()
  Makes a widget invisible.

• Fl_Image *image ()
  Gets the image that is used as part of the widget label when in the active state.

• const Fl_Image *image () const
  Gets the image that is used as part of the widget label when in the active state.

• void image (Fl_Image &img)
Sets the image to use as part of the widget label when in the active state.

- **void image (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const**
  Returns whether the image is managed by the widget.

- **int inside (const Fl_Widget *wgt) const**
  Checks if this widget is a child of wgt.

- **int is_label_copied () const**
  Returns whether the current label was assigned with copy_label().

- **const char * label () const**
  Gets the current label text.

- **void label (const char *text)**
  Sets the current label pointer.

- **void label (Fl_Labeltype a, const char *b)**
  Shortcut to set the label text and type in one call.

- **Fl_Color labelcolor () const**
  Gets the label color.

- **void labelcolor (Fl_Color c)**
  Sets the label color.

- **Fl_Font labelfont () const**
  Gets the font to use.

- **void labelfont (Fl_Font f)**
  Sets the font to use.

- **Fl_Fontsize labelsize () const**
  Gets the font size in pixels.

- **void labelsize (Fl_Fontsize pix)**
  Sets the font size in pixels.

- **Fl_Labeltype labeltype () const**
  Gets the label type.

- **void labeltype (Fl_Labeltype a)**
  Sets the label type.

- **void measure_label (int &ww, int &hh) const**
  Sets width ww and height hh accordingly with the label size.

- **bool needs_keyboard () const**
  Returns whether this widget needs a keyboard.

- **void needs_keyboard (bool needs)**
  Sets whether this widget needs a keyboard.

- **unsigned int output () const**
  Returns if a widget is used for output only.

- **Fl_Group * parent () const**
  Returns a pointer to the parent widget.

- **void parent (Fl_Group *p)**
  Internal use only - "for hacks only".

- **void position (int X, int Y)**
  Repositions the window or widget.

- **void redraw ()**
  Schedules the drawing of the widget.

- **void redraw_label ()**
  Schedules the drawing of the label.

- **Fl_Color selection_color () const**
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
  • int shortcut_label () const
    Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void show ()
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.
• const char ∗tooltip () const
  Gets the current tooltip text.
• void tooltip (const char ∗text)
  Sets the current tooltip text.
• Fl_Window ∗top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window ∗top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void ∗user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void ∗v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
Class Documentation

Modifies keyboard focus navigation.

- int visible_r () const
  Returns whether a widget and all its parents are visible.

- int w () const
  Gets the widget width.

- FL_When when () const
  Returns the conditions under which the callback is called.

- void when (uchar i)
  Sets the flags used to decide when a callback is called.

- Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

- int x () const
  Gets the widget position in its window.

- int y () const
  Gets the widget position in its window.

- virtual ~Fl_Widget ()
  Destroys the widget.

Protected Member Functions

- void draw () FL_OVERRIDE
  Draws the file input widget.

Protected Member Functions inherited from Fl_Input

- void draw () FL_OVERRIDE
  Draws the widget.

- int handle_key ()
  Handles a keystroke.

- int handle_rmb ()
  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

- int apply_undo ()
  Apply the current undo/redo operation.

- void drawtext (int, int, int, int)
  Draws the text in the passed bounding box.

- void drawtext (int, int, int, int, bool draw_active)
  Draws the text in the passed bounding box.

- void handle_mouse (int, int, int, int, int keepmark=0)
  Handles mouse clicks and mouse moves.

- int handletext (int e, int, int, int, int)
  Handles all kinds of text field related events.

- int line_end (int i) const
  Finds the end of a line.

- int line_start (int i) const
  Finds the start of a line.

- int linesPerPage ()

- void maybe_do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

- int up_down_position (int, int keepmark=0)
  Moves the cursor to the column given by up_down_pos.
• int word_end (int i) const  
  Finds the end of a word.
• int word_start (int i) const  
  Finds the start of a word.
• int xscroll () const
• int yscroll () const
• void yscroll (int yOffset)

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)  
  Clears a flag in the flags mask.
• void draw_backdrop () const  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const  
  Draws the widget box according its box style.
• void draw_box (Fl_Boxtype t, Fl_Color c) const  
  Draws a box of type t, of color c at the widget's position and size.
• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const  
  Draws a box of type t, of color c at the position X,Y and size W,H.
• void draw_focus () const  
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int X, int Y , int W, int H) const  
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const  
  Draws a focus box for the widget at the given position and size.
• void draw_label () const  
  Draws the widget's label at the defined label position.
• void draw_label (int, int, int, int) const  
  Draws the label in an arbitrary bounding box.
• Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)  
  Creates a widget at the given position and size.
• unsigned int flags () const  
  Gets the widget flags mask.
• void h (int v)  
  Internal use only.
• void set_flag (unsigned int c)  
  Sets a flag in the flags mask.
• void w (int v)  
  Internal use only.
• void x (int v)  
  Internal use only.
• void y (int v)  
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget ∗widget, void ∗data)  
  The default callback for all widgets that don't set a callback.
• static unsigned int label_shortcut (const char ∗t)  
  Returns the Unicode value of the '&x' shortcut in a given text.
• static int test_shortcut (const char ∗t, const bool require_alt=false)  
  Returns true if the given text t contains the entered 'x' shortcut.
Static Public Attributes inherited from Fl_Input

- static const char ∗ copy_menu_text = "Copy"
  [this text may be customized at run-time]
- static const char ∗ cut_menu_text = "Cut"
  [this text may be customized at run-time]
- static const char ∗ paste_menu_text = "Paste"
  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLtip_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCEPTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

12.38.1 Detailed Description

This widget displays a pathname in a text input field. A navigation bar located above the input field allows the user to navigate upward in the directory tree. You may want to handle FL_WHEN_CHANGED events for tracking text changes and also FL_WHEN_RELEASE for button release when changing to parent dir. FL_WHEN_RELEASE callback won't be called if the directory clicked is the same as the current one.

![Figure 12.17 Fl_File_Input](image)

Note

As all Fl_Input derived objects, Fl_File_Input may call its callback when losing focus (see FL_UNFOCUS) to update its state like its cursor shape. One resulting side effect is that you should call clear_changed() early in your callback to avoid reentrant calls if you plan to show another window or dialog box in the callback.

12.38.2 Constructor & Destructor Documentation

12.38.2.1 Fl_File_Input()

Fl_File_Input::Fl_File_Input ( int X, int Y, int W, int H, const char ∗ L = 0 )

Creates a new Fl_File_Input widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X, Y, W, H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

### 12.38.3 Member Function Documentation

#### 12.38.3.1 down_box()

```cpp
default void Fl_File_Input::down_box (Fl_Boxtype b) [inline]
```
Sets the box type to use for the navigation bar.

#### 12.38.3.2 draw()

```cpp
default void Fl_File_Input::draw () [protected], [virtual]
```
Draws the file input widget.
Implements Fl_Widget.

#### 12.38.3.3 errorcolor() [1/2]

```cpp
Fl_Color Fl_File_Input::errorcolor () const [inline]
```
Gets the current error color.
Returns FL_RED since FLTK 1.4.0 (default in 1.3.x). Retained for backwards compatibility.

**Deprecated** Will be removed in FLTK 1.5.0 or higher.

**Todo** Remove Fl_File_Input::errorcolor() in FLTK 1.5.0 or higher.

#### 12.38.3.4 errorcolor() [2/2]

```cpp
void Fl_File_Input::errorcolor (Fl_Color c) [inline]
```
Sets the current error color to c.
Does nothing since FLTK 1.4.0. Retained for backwards compatibility.

**Deprecated** Will be removed in FLTK 1.5.0 or higher.

**Todo** Remove Fl_File_Input::errorcolor(Fl_Color) in FLTK 1.5.0 or higher.

#### 12.38.3.5 handle()

```cpp
int Fl_File_Input::handle (int event) [virtual]
```
Handle events in the widget.
Return non zero if event is handled.

**Parameters**

| in  | event |

Reimplemented from Fl_Widget.
12.38.3.6 value() [1/2]

int Fl_File_Input::value (const char * str)

Sets the value of the widget given a new string value.
Returns non 0 on success.

Parameters

| in  | str  | new string value |

12.38.3.7 value() [2/2]

int Fl_File_Input::value (const char * str, int len)

Sets the value of the widget given a new string value and its length.
Returns non 0 on success.

Parameters

| in  | str  | new string value |
| in  | len  | length of value |

The documentation for this class was generated from the following files:

- Fl_File_Input.H
- Fl_File_Input.cxx

12.39 Fl_Fill_Dial Class Reference

Draws a dial with a filled arc.

#include <Fl_Fill_Dial.H>

Inheritance diagram for Fl_Fill_Dial:

```
    Fl_Widget
     ↓
    Fl_Valuator
     ↓
    Fl_Dial
     ↓
Fl_Fill_Dial
```

Public Member Functions

- **Fl_Fill_Dial** (int X, int Y, int W, int H, const char *L)
  
  Creates a filled dial, also setting its type to FL_FILL_DIAL.

Public Member Functions inherited from Fl_Dial

- short **angle1** () const
  
  Sets Or gets the angles used for the minimum and maximum values.

- void **angle1** (short a)
See short `angle1() const`.

- **short angle2() const**
  
  See short `angle1() const`.

- **void angle2(short a)**
  
  See short `angle1() const`.

- **void angles(short a, short b)**
  
  See short `angle1() const`.

- **Fl_Dial(int x, int y, int w, int h, const char ∗l=0)**
  
  Creates a new `Fl_Dial` widget using the given position, size, and label string.

- **int handle(int) FL_OVERRIDE**
  
  Allow subclasses to handle event based on current position and size.

### Public Member Functions inherited from `Fl_Valuator`

- **void bounds(double a, double b)**
  
  Sets the minimum (a) and maximum (b) values for the valuator widget.

- **double clamp(double)**
  
  Clamps the passed value to the valuator range.

- **virtual int format(char ∗)**
  
  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.

- **double increment(double, int)**
  
  Adds n times the step value to the passed value.

- **double maximum() const**
  
  Gets the maximum value for the valuator.

- **void maximum(double a)**
  
  Sets the maximum value for the valuator.

- **double minimum() const**
  
  Gets the minimum value for the valuator.

- **void minimum(double a)**
  
  Sets the minimum value for the valuator.

- **void precision(int digits)**
  
  Sets the step value to \(1.0 / 10^{\text{digits}}\).

- **void range(double a, double b)**
  
  Sets the minimum and maximum values for the valuator.

- **double round(double)**
  
  Round the passed value to the nearest step increment.

- **double step() const**
  
  Gets or sets the step value.

- **void step(double a, int b)**
  
  See double `Fl_Valuator::step() const`

- **void step(double s)**
  
  See double `Fl_Valuator::step() const`.

- **void step(int a)**
  
  See double `Fl_Valuator::step() const`.

- **double value() const**
  
  Gets the floating point(double) value.

- **int value(double)**
  
  Sets the current value.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * as_gl_window () const
- virtual Fl_Group * as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * as_group () const
- virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * as_window () const
- void bind_deimage (Fl_Image * img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image * img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback * cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback * cb, Fl_Callback_User_Data * p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback * cb, void * p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 * cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 * cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
Checks if the widget value changed since the last callback.

• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

• void clear_changed ()
  Marks the value of the widget as unchanged.

• void clear_damage (uchar c=0)
  Clears or sets the damage flags.

• void clear_output ()
  Sets a widget to accept input.

• void clear_visible ()
  Hides the widget.

• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

• Fl_Color color () const
  Gets the background color of the widget.

• void color (Fl_Color bg)
  Sets the background color of the widget.

• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

• Fl_Color color2 () const
  For back compatibility only.

• void color2 (unsigned a)
  For back compatibility only.

• int contains (const Fl_Widget ∗w) const
  Checks if w is a child of this widget.

• void copy_label (const char ∗new_label)
  Sets the current label.

• void copy_tooltip (const char ∗text)
  Sets the current tooltip text.

• uchar damage () const
  Returns non-zero if draw() needs to be called.

• void damage (uchar c)
  Sets the damage bits for the widget.

• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

• int damage_resize (int, int, int, int)
  Internal use only.

• void deactivate ()
  Deactivates the widget.

• Fl_Image ∗deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

• const Fl_Image ∗deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

• void deimage (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the inactive state.

• void deimage (Fl_Image ∗&img)
  Sets the image to use as part of the widget label when in the inactive state.

• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void **do_callback** (**Fl_Widget** ∗**widget**, long **arg**, **Fl_Callback_Reason** **reason**=**FL_REASON_UNKNOWN**)  
  *Calls the widget callback function with arbitrary arguments.*

• void **do_callback** (**Fl_Widget** ∗**widget**, void ∗**arg**=0, **Fl_Callback_Reason** **reason**=**FL_REASON_UNKNOWN**)  
  *Calls the widget callback function with arbitrary arguments.*

• void **draw_label** (int, int, int, int, **Fl_Align**) const  
  *Draws the label in an arbitrary bounding box with an arbitrary alignment.*

• int **h** () const  
  *Gets the widget height.*

• virtual void **hide** ()  
  *Makes a widget invisible.*

• **Fl_Image** ∗**image** ()  
  *Gets the image that is used as part of the widget label when in the active state.*

• const **Fl_Image** ∗**image** () const  
  *Gets the image that is used as part of the widget label when in the active state.*

• void **image** (**Fl_Image** &**img**)  
  *Sets the image to use as part of the widget label when in the active state.*

• void **image** (**Fl_Image** ∗**img**)  
  *Sets the image to use as part of the widget label when in the active state.*

• int **image_bound** () const  
  *Returns whether the image is managed by the widget.*

• int **inside** (**const Fl_Widget** ∗**wgt**) const  
  *Checks if this widget is a child of wgt.*

• int **is_label_copied** () const  
  *Returns whether the current label was assigned with copy_label().*

• const char ∗**label** () const  
  *Gets the current label text.*

• void **label** (const char ∗**text**)  
  *Sets the current label pointer.*

• void **label** (**Fl_Labeltype** **a**, const char ∗**b**)  
  *Shortcut to set the label text and type in one call.*

• **Fl_Color** **labelcolor** () const  
  *Gets the label color.*

• void **labelcolor** (**Fl_Color** **c**)  
  *Sets the label color.*

• **Fl_Font** **labelfont** () const  
  *Gets the font to use.*

• void **labelfont** (**Fl_Font** **f**)  
  *Sets the font to use.*

• **Fl_Fontsize** **labelsize** () const  
  *Gets the font size in pixels.*

• void **labelsize** (**Fl_Fontsize** **pix**)  
  *Sets the font size in pixels.*

• **Fl_Labeltype** **labeltype** () const  
  *Gets the label type.*

• void **labeltype** (**Fl_Labeltype** **a**)  
  *Sets the label type.*

• void **measure_label** (int &**ww**, int &**hh**) const  
  *Sets width ww and height hh accordingly with the label size.*

• bool **needs_keyboard** () const  
  *Returns whether this widget needs a keyboard.*

• void **needs_keyboard** (bool **needs**)
Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.

• Fl_Group ∗ parent () const
  Returns a pointer to the parent widget.

• void parent (Fl_Group ∗p)
  Internal use only - "for hacks only".

• void position (int X, int Y)
  Repositions the window or widget.

• void redraw ()
  Schedules the drawing of the widget.

• void redraw_label ()
  Schedules the drawing of the label.

• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.

• Fl_Color selection_color () const
  Gets the selection color.

• void selection_color (Fl_Color a)
  Sets the selection color.

• void set_active ()
  Marks the widget as active without sending events or changing focus.

• void set_changed ()
  Marks the value of the widget as changed.

• void set_output ()
  Sets a widget to output only.

• void set_visible ()
  Makes the widget visible.

• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.

• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.

• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

• virtual void show ()
  Makes a widget visible.

• void size (int W, int H)
  Changes the size of the widget.

• int take_focus ()
  Gives the widget the keyboard focus.

• unsigned int takesevents () const
  Returns if the widget is able to take events.

• int testShortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.

• const char ∗ tooltip () const
  Gets the current tooltip text.

• void tooltip (const char ∗text)
  Sets the current tooltip text.

• Fl_Window ∗ top_window () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• **uchar type () const**
  
  Gets the widget type.

• **void type (uchar t)**
  
  Sets the widget type.

• **int use_accents_menu ()**
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• **void * user_data () const**
  
  Gets the user data for this widget.

• **void user_data (Fl_Callback_User_Data *v, bool auto_free)**
  
  Sets the user data for this widget.

• **void user_data (void *v)**
  
  Sets the user data for this widget.

• **unsigned int visible () const**
  
  Returns whether a widget is visible.

• **unsigned int visible_focus () const**
  
  Checks whether this widget has a visible focus.

• **void visible_focus (int v)**
  
  Modifies keyboard focus navigation.

• **int visible_r () const**
  
  Returns whether a widget and all its parents are visible.

• **int w () const**
  
  Gets the widget width.

• **FL_When when () const**
  
  Returns the conditions under which the callback is called.

• **void when (uchar i)**
  
  Sets the flags used to decide when a callback is called.

• **FL_Window * window () const**
  
  Returns a pointer to the nearest parent window up the widget hierarchy.

• **int x () const**
  
  Gets the widget position in its window.

• **int y () const**
  
  Gets the widget position in its window.

• **virtual ~Fl_Widget ()**
  
  Destroys the widget.

### Additional Inherited Members

**Static Public Member Functions inherited from Fl_Widget**

• **static void default_callback (Fl_Widget *widget, void *data)**
  
  The default callback for all widgets that don't set a callback.

• **static unsigned int label_shortcut (const char *t)**
  
  Returns the Unicode value of the '&x' shortcut in a given text.

• **static int test_shortcut (const char *, const bool require_alt=false)**
  
  Returns true if the given text t contains the entered '&x' shortcut.
Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6,
  CHANGED = 1<<7, OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9,
  COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTip_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTip = 1<<17, FULLSCREEN = 1<<18,
  MAC_USE_ACCENTS_MENU = 1<<19, NEEDS_KEYBOARD = 1<<20,
  IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22, AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<25, USERFLAG2 = 1<<26, USERFLAG1 = 1<<27
}

flags possible values enumeration.

Protected Member Functions inherited from Fl_Dial

- void draw () FL_OVERRIDE
  Draws dial at current position and size.
- void draw (int X, int Y, int W, int H)
  Draws dial at given position and size.
- void handle (int event, int X, int Y, int W, int H)
  Allows subclasses to handle event based on given position and size.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char ∗L)
  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void handle_drag (double newvalue)
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void handle_push ()
  Stores the current value in the previous value.
- void handle_release ()
  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int horizontal () const
  Tells if the valuator is an FL_HORIZONTAL one.
- double previous_value () const
  Gets the previous floating point value before an event changed it.
- void set_value (double v)
  Sets the current floating point value.
- double softclamp (double)
  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()
  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
Draws the widget box according its box style.

- `void draw_box (Fl_Boxtype t, Fl_Color c) const`
  Draws a box of type t, of color c at the widget's position and size.

- `void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const`
  Draws a box of type t, of color c at the position X,Y and size W,H.

- `void draw_focus () const`
  Draws a focus rectangle around the widget.

- `void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const`
  Draws a focus rectangle around the widget.

- `void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const`
  Draws a focus box for the widget at the given position and size.

- `void draw_label () const`
  Draws the widget's label at the defined label position.

- `void draw_label (int, int, int, int) const`
  Draws the label in an arbitrary bounding box.

- `Fl_Widget (int x, int y, int w, int h, const char *label=0L)`
  Creates a widget at the given position and size.

- `unsigned int flags () const`
  Gets the widget flags mask.

- `void h (int v)`
  Internal use only.

- `void set_flag (unsigned int c)`
  Sets a flag in the flags mask.

- `void w (int v)`
  Internal use only.

- `void x (int v)`
  Internal use only.

- `void y (int v)`
  Internal use only.

### 12.39.1 Detailed Description

Draws a dial with a filled arc.

The documentation for this class was generated from the following files:

- `Fl_Fill_Dial.H`
- `Fl_Fill_Dial.cxx`

### 12.40 Fl_Fill_Slider Class Reference

Widget that draws a filled horizontal slider, useful as a progress or value meter.

```c
#include <Fl_Fill_Slider.H>
```

Inheritance diagram for Fl_Fill_Slider:

```
Fl_Widget
    ↓
Fl_Valuator
    ↓
Fl_Slider
    ↓
Fl_Fill_Slider
```
Public Member Functions

- **Fl_Fill_Slider** (int X, int Y, int W, int H, const char *L=0)
  
  Creates the slider from its position, size and optional title.

Public Member Functions inherited from **Fl_Slider**

- void **bounds** (double a, double b)
  
  Sets the minimum (a) and maximum (b) values for the valuator widget.

- **Fl_Slider** (int X, int Y, int W, int H, const char *L=0)
  
  Creates a new **Fl_Slider** widget using the given position, size, and label string.

- **Fl_Slider** (uchar t, int X, int Y, int W, int H, const char *L)
  
  Creates a new **Fl_Slider** widget using the given type, position, size, and label string.

- int **handle** (int) FL_OVERRIDE
  
  Handles the specified event.

- int **scrollvalue** (int pos, int size, int first, int total)
  
  Sets the size and position of the sliding knob in the box.

- **Fl_Boxtype** **slider** () const
  
  Gets the slider box type.

- void **slider** (Fl_Boxtype c)
  
  Sets the slider box type.

- float **slider_size** () const
  
  Get the dimensions of the moving piece of slider.

- void **slider_size** (double v)
  
  Set the dimensions of the moving piece of slider.

Public Member Functions inherited from **Fl_Valuator**

- void **bounds** (double a, double b)
  
  Sets the minimum (a) and maximum (b) values for the valuator widget.

- double **clamp** (double)
  
  Clamps the passed value to the valuator range.

- virtual int **format** (char *p)
  
  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.

- double **increment** (double, int)
  
  Adds n times the step value to the passed value.

- double **maximum** () const
  
  Gets the maximum value for the valuator.

- void **maximum** (double a)
  
  Sets the maximum value for the valuator.

- double **minimum** () const
  
  Gets the minimum value for the valuator.

- void **minimum** (double a)
  
  Sets the minimum value for the valuator.

- void **precision** (int digits)
  
  Sets the step value to \(1.0 / 10^{{\text{digits}}}\).

- void **range** (double a, double b)
  
  Sets the minimum and maximum values for the valuator.

- double **round** (double)
  
  Round the passed value to the nearest step increment.

- double **step** () const
  
  Gets or sets the step value.
• void **step** (double a, int b)
  
    See double Fl_Valuator::step() const

• void **step** (double s)
  
    See double Fl_Valuator::step() const.

• void **step** (int a)
  
    See double Fl_Valuator::step() const

• double **value** () const
  
    Gets the floating point(double) value.

• int **value** (double)
  
    Sets the current value.

**Public Member Functions inherited from Fl_Widget**

• void **_clear_fullscreen** ()

• void **_set_fullscreen** ()

• void **activate** ()
  
    Activates the widget.

• unsigned int **active** () const
  
    Returns whether the widget is active.

• int **active_r** () const
  
    Returns whether the widget and all of its parents are active.

• Fl_Align **align** () const
  
    Gets the label alignment.

• void **align** (Fl_Align alignment)
  
    Sets the label alignment.

• long **argument** () const
  
    Gets the current user data (long) argument that is passed to the callback function.

• void **argument** (long v)
  
    Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window * **as_gl_window** ()
  
    Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const * **as_gl_window** () const

• virtual Fl_Group * **as_group** ()
  
    Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Group const * **as_group** () const

• virtual Fl_Window * **as_window** ()
  
    Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window const * **as_window** () const

• void **bind_deimage** (Fl_Image * img)
  
    Sets the image to use as part of the widget label when in the inactive state.

• void **bind_deimage** (int f)
  
    Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

• void **bind_image** (Fl_Image * img)
  
    Sets the image to use as part of the widget label when in the active state.

• void **bind_image** (int f)
  
    Bind the image to the widget, so the widget will delete the image when it is no longer needed.

• Fl_Boxtype **box** () const
  
    Gets the box type of the widget.

• void **box** (Fl_Boxtype new_box)
  
    Sets the box type for the widget.
• **Fl_Callback_p callback () const**
  
  Gets the current callback function for the widget.

• **void callback (Fl_Callback *cb)**
  
  Sets the current callback function for the widget.

• **void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)**
  
  Sets the current callback function and managed user data for the widget.

• **void callback (Fl_Callback *cb, void *p)**
  
  Sets the current callback function and data for the widget.

• **void callback (Fl_Callback0 *cb)**
  
  Sets the current callback function for the widget.

• **void callback (Fl_Callback1 *cb, long p=0)**
  
  Sets the current callback function for the widget.

• **unsigned int changed () const**
  
  Checks if the widget value changed since the last callback.

• **void clear_active ()**
  
  Marks the widget as inactive without sending events or changing focus.

• **void clear_changed ()**
  
  Marks the value of the widget as unchanged.

• **void clear_damage (uchar c=0)**
  
  Clears or sets the damage flags.

• **void clear_output ()**
  
  Sets a widget to accept input.

• **void clear_visible ()**
  
  Hides the widget.

• **void clear_visible_focus ()**
  
  Disables keyboard focus navigation with this widget.

• **Fl_Color color () const**
  
  Gets the background color of the widget.

• **void color (Fl_Color bg)**
  
  Sets the background color of the widget.

• **void color (Fl_Color bg, Fl_Color sel)**
  
  Sets the background and selection color of the widget.

• **Fl_Color color2 () const**
  
  For back compatibility only.

• **void color2 (unsigned a)**
  
  For back compatibility only.

• **int contains (const Fl_Widget *w) const**
  
  Checks if w is a child of this widget.

• **void copy_label (const char *new_label)**
  
  Sets the current label.

• **void copy_tooltip (const char *text)**
  
  Sets the current tooltip text.

• **uchar damage () const**
  
  Returns non-zero if draw() needs to be called.

• **void damage (uchar c)**
  
  Sets the damage bits for the widget.

• **void damage (uchar c, int x, int y, int w, int h)**
  
  Sets the damage bits for an area inside the widget.

• **int damage_resize (int, int, int)**
  
  Internal use only.

• **void deactivate ()**

Generated by Doxygen
Deactivates the widget.

- FL_Image * deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

- const FL_Image * deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

- void deimage (FL_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

- void deimage (FL_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

- void do_callback (FL_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

- void do_callback (FL_Widget *widget, long arg, FL_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- void do_callback (FL_Widget *widget, void *arg=0, FL_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- int h () const
  Gets the widget height.

- virtual void hide ()
  Makes a widget invisible.

- FL_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.

- const FL_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.

- void image (FL_Image &img)
  Sets the image to use as part of the widget label when in the active state.

- void image (FL_Image *img)
  Sets the image to use as part of the widget label when in the active state.

- int image_bound () const
  Returns whether the image is managed by the widget.

- int inside (const FL_Widget *wgt) const
  Checks if this widget is a child of wgt.

- int is_labelCopied () const
  Returns whether the current label was assigned with copy_label().

- const char * label () const
  Gets the current label text.

- void label (const char *text)
  Sets the current label pointer.

- void label (FL_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.

- FL_Color labelcolor () const
  Gets the label color.

- void labelcolor (FL_Color c)
  Sets the label color.

- FL_Font labelfont () const
  Gets the font to use.

- void labelfont (FL_Font f)
  Sets the font to use.
- **Fl_Fontsize labelsize () const**
  Gets the font size in pixels.

- **void labelsize (Fl_Fontsize pix)**
  Sets the font size in pixels.

- **Fl_Labeltype labeltype () const**
  Gets the label type.

- **void labeltype (Fl_Labeltype a)**
  Sets the label type.

- **void measure_label (int &ww, int &hh) const**
  Sets width ww and height hh accordingly with the label size.

- **bool needs_keyboard () const**
  Returns whether this widget needs a keyboard.

- **void needs_keyboard (bool needs)**
  Sets whether this widget needs a keyboard.

- **unsigned int output () const**
  Returns if a widget is used for output only.

- **Fl_Group * parent () const**
  Returns a pointer to the parent widget.

- **void parent (Fl_Group *p)**
  Internal use only - "for hacks only".

- **void position (int X, int Y)**
  Repositions the window or widget.

- **void redraw ()**
  Schedules the drawing of the widget.

- **void redraw_label ()**
  Schedules the drawing of the label.

- **virtual void resize (int x, int y, int w, int h)**
  Changes the size or position of the widget.

- **Fl_Color selection_color () const**
  Gets the selection color.

- **void selection_color (Fl_Color a)**
  Sets the selection color.

- **void set_active ()**
  Marks the widget as active without sending events or changing focus.

- **void set_changed ()**
  Marks the value of the widget as changed.

- **void set_output ()**
  Sets a widget to output only.

- **void set_visible ()**
  Makes the widget visible.

- **void set_visible_focus ()**
  Enables keyboard focus navigation with this widget.

- **int shortcut_label () const**
  Returns whether the widget's label uses ' & ' to indicate shortcuts.

- **void shortcut_label (int value)**
  Sets whether the widget's label uses ' & ' to indicate shortcuts.

- **virtual void show ()**
  Makes a widget visible.

- **void size (int W, int H)**
  Changes the size of the widget.

- **int take_focus ()**
Gives the widget the keyboard focus.

• unsigned int takesevents() const
  Returns if the widget is able to take events.

• int test_shortcut()
  Returns true if the widget's label contains the entered 'x' shortcut.

• const char * tooltip() const
  Gets the current tooltip text.

• void tooltip(const char *text)
  Sets the current tooltip text.

• Fl_Window * top_window() const
  Returns a pointer to the top-level window for the widget.

• Fl_Window * top_window_offset(int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type() const
  Gets the widget type.

• void type(uchar t)
  Sets the widget type.

• int use_accents_menu()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void * user_data() const
  Gets the user data for this widget.

• void user_data(Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

• void user_data(void *v)
  Sets the user data for this widget.

• unsigned int visible() const
  Returns whether a widget is visible.

• unsigned int visible_focus() const
  Checks whether this widget has a visible focus.

• void visible_focus(int v)
  Modifies keyboard focus navigation.

• int visible_r() const
  Returns whether a widget and all its parents are visible.

• int w() const
  Gets the widget width.

• Fl_When when() const
  Returns the conditions under which the callback is called.

• void when(uchar i)
  Sets the flags used to decide when a callback is called.

• Fl_Window * window() const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x() const
  Gets the widget position in its window.

• int y() const
  Gets the widget position in its window.

• virtual ~Fl_Widget()
  Destroys the widget.
Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don’t set a callback.
- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the ‘&x’ shortcut in a given text.
- static int test_shortcut (const char *t, const bool require_alt=false)
  Returns true if the given text t contains the entered ‘&x’ shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Slider

- void draw () FL_OVERRIDE
  Draws the widget.
- void draw (int, int, int, int)
- int handle (int, int, int, int)

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y , int W, int H, const char *L)
  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void handle_drag (double newvalue)
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void handle_push ()
  Stores the current value in the previous value.
- void handle_release ()
  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int horizontal () const
  Tells if the valuator is an FL_HORIZONTAL one.
- double previous_value () const
  Gets the previous floating point value before an event changed it.
- void set_value (double v)
  Sets the current floating point value.
- double softclamp (double)
  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()
  Asks for partial redraw.

Generated by Doxygen
Protected Member Functions inherited from Fl_Widget

- void clear_flag(unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop() const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box() const
  Draws the widget box according its box style.
- void draw_box(Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus() const
  Draws a focus rectangle around the widget.
- void draw_focus(Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
- void draw_focus(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void draw_label() const
  Draws the widget's label at the defined label position.
- void draw_label(int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget(int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.
- unsigned int flags() const
  Gets the widget flags mask.
- void h(int v)
  Internal use only.
- void set_flag(unsigned int c)
  Sets a flag in the flags mask.
- void w(int v)
  Internal use only.
- void x(int v)
  Internal use only.
- void y(int v)
  Internal use only.

12.40.1 Detailed Description

Widget that draws a filled horizontal slider, useful as a progress or value meter.
The documentation for this class was generated from the following files:

- Fl_Fill_Slider.H
- Fl_Slider.cxx

12.41 Fl_Flex Class Reference

Fl_Flex is a container (layout) widget for one row or one column of widgets.
#include <Fl_Flex.H>
Inheritance diagram for Fl_Flex:
Public Types

- `enum { VERTICAL = 0 , HORIZONTAL = 1 , COLUMN = 0 , ROW = 1 }`

Public Member Functions

- `virtual void end ()`
  
  Ends automatic child addition and resizes all children.

- `void fixed (Fl_Widget &w, int size)`
  
  Set the horizontal or vertical size of a child widget.

- `int fixed (Fl_Widget *w) const`
  
  Return whether the given widget has a fixed size or resizes dynamically.

- `void fixed (Fl_Widget *w, int size)`
  
  Set the horizontal or vertical size of a child widget.

- `Fl_Flex (int direction)`
  
  Construct a new Fl_Flex widget specifying its layout.

- `Fl_Flex (int w, int h, int direction)`
  
  Construct a new Fl_Flex widget specifying its layout and size.

- `Fl_Flex (int X, int Y, int W, int H, const char *L=0)`
  
  Construct a new Fl_Flex widget with the given position, size, and label.

- `Fl_Flex (int x, int y, int w, int h, int direction)`
  
  Construct a new Fl_Flex widget specifying its layout, position, and size.

- `int gap () const`
  
  Return the gap size of the widget.

- `void gap (int g)`
  
  Set the gap size of the widget.

- `int horizontal () const`
  
  Returns non-zero (true) if Fl_Flex alignment is horizontal (row mode).

- `void layout ()`
  
  Calculates the layout of the widget and redraws it.

- `int margin () const`
  
  Returns the left margin size of the widget.

- `int margin (int left, int top, int right, int bottom) const`
  
  Returns all (four) margin sizes of the widget.

- `void margin (int left, int top, int right, int bottom)`
  
  Set the margin sizes at all four edges of the Fl_Flex widget.

- `void margin (int m, int g=-1)`
  
  Set the margin and optionally the gap size of the widget.

- `bool need_layout () const`
  
  Returns whether layout calculation is required.

- `void need_layout (int set)`
  
  Set or reset the request to calculate the layout of children.

- `void resize (int x, int y, int w, int h) FL_OVERRIDE`
  
  Resize the container and calculate all child positions and sizes.
• int spacing () const
  Gets the number of extra pixels of blank space that are added between the children.
• void spacing (int i)
  Sets the number of extra pixels of blank space that are added between the children.

Public Member Functions inherited from Fl_Group

• FL_Widget *&_ddfdesign_kludge ()
  This is for forms compatibility only.
• void add (FL_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.
• void add (FL_Widget *)
  See void Fl_Group::add(FL_Widget &w)
• void add_resizable (FL_Widget &o)
  Adds a widget to the group and makes it the resizable widget.
• FL_Widget *const * array () const
  Returns a pointer to the array of children.
• FL_Group const * as_group () const FL_OVERRIDE
• FL_Group * as_group () FL_OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• void begin ()
  Sets the current group so you can build the widget tree by just constructing the widgets.
• FL_Widget * child (int n) const
  Returns array()[n].
• int children () const
  Returns how many child widgets the group has.
• void clear ()
  Deletes all child widgets from memory recursively.
• unsigned int clip_children ()
  Returns the current clipping mode.
• void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.
• virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.
• void end ()
  Exactly the same as current(this->parent()).
• int find (const Fl_Widget &o) const
  See int Fl_Group::find(const Fl_Widget *w) const.
• int find (const Fl_Widget *) const
  Searches the child array for the widget and returns the index.
• FL_Group (int, int, int, int, const char * = 0)
  Creates a new Fl_Group widget using the given position, size, and label string.
• void focus (FL_Widget *)
• void forms_end ()
  This is for forms compatibility only.
• int handle (int) FL_OVERRIDE
  Handles the specified event.
• void init_sizes ()
  Resets the internal array of widget sizes and positions.
• void insert (FL_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)
  This does insert(w, find(before)).
- void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)
  Removes the widget o from the group.
- void remove (int index)
  Removes the widget at index from the group but does not delete it.
- Fl_Widget * resizable () const
  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)
  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- void resize (int, int, int, int) FL_OVERRIDE
  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()
  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * as_gl_window () const
- virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * as_window () const
- void bind_deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• **Fl_Boxtype box () const**
  Gets the box type of the widget.

• **void box (Fl_Boxtype new_box)**
  Sets the box type for the widget.

• **Fl_Callback_p callback () const**
  Gets the current callback function for the widget.

• **void callback (Fl_Callback ∗cb)**
  Sets the current callback function for the widget.

• **void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)**
  Sets the current callback function and managed user data for the widget.

• **void callback (Fl_Callback ∗cb, void ∗p)**
  Sets the current callback function and data for the widget.

• **void callback (Fl_Callback0 ∗cb)**
  Sets the current callback function for the widget.

• **void callback (Fl_Callback1 ∗cb, long p=0)**
  Sets the current callback function for the widget.

• **unsigned int changed () const**
  Checks if the widget value changed since the last callback.

• **void clear_active ()**
  Marks the widget as inactive without sending events or changing focus.

• **void clear_changed ()**
  Marks the value of the widget as unchanged.

• **void clear_damage (uchar c=0)**
  Clears or sets the damage flags.

• **void clear_output ()**
  Sets a widget to accept input.

• **void clear_visible ()**
  Hides the widget.

• **void clear_visible_focus ()**
  Disables keyboard focus navigation with this widget.

• **Fl_Color color () const**
  Gets the background color of the widget.

• **void color (Fl_Color bg)**
  Sets the background color of the widget.

• **void color (Fl_Color bg, Fl_Color sel)**
  Sets the background and selection color of the widget.

• **Fl_Color color2 () const**
  For back compatibility only.

• **void color2 (unsigned a)**
  For back compatibility only.

• **int contains (const Fl_Widget ∗w) const**
  Checks if w is a child of this widget.

• **void copy_label (const char ∗new_label)**
  Sets the current label.

• **void copy_tooltip (const char ∗text)**
  Sets the current tooltip text.

• **uchar damage () const**
  Returns non-zero if draw() needs to be called.

• **void damage (uchar c)**
  Sets the damage bits for the widget.

• **void damage (uchar c, int x, int y, int w, int h)**
  For back compatibility only.
Setting the damage bits for an area inside the widget.

- `int damage_resize(int, int, int, int)`
  
  Internal use only.

- `void deactivate()`
  
  Deactivates the widget.

- `void *deimage()`
  
  Gets the image that is used as part of the widget label when in the inactive state.

  ```
  const Fl_Image *deimage() const
  
  Gets the image that is used as part of the widget label when in the inactive state.
  ```

  ```
  void deimage(Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the inactive state.
  ```

  ```
  void deimage(Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the inactive state.
  ```

  ```
  int deimage_bound() const
  
  Returns whether the inactive image is managed by the widget.
  ```

- `void do_callback(Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`

  Calls the widget callback function with default arguments.

- `void do_callback(Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`

  Calls the widget callback function with arbitrary arguments.

- `void do_callback(Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`

  Calls the widget callback function with arbitrary arguments.

- `void draw_label(int, int, int, int, Fl_Align) const`

  Draws the label in an arbitrary bounding box with an arbitrary alignment.

  ```
  int h() const
  
  Gets the widget height.
  ```

  ```
  virtual void hide() const
  
  Makes a widget invisible.
  ```

- `Fl_Image *image()`

  Gets the image that is used as part of the widget label when in the active state.

  ```
  const Fl_Image *image() const
  
  Gets the image that is used as part of the widget label when in the active state.
  ```

  ```
  void image(Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the active state.
  ```

  ```
  void image(Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the active state.
  ```

  ```
  int image_bound() const
  
  Returns whether the image is managed by the widget.
  ```

- `int inside(const Fl_Widget *wgt) const`

  Checks if this widget is a child of `wgt`.

- `int is_label_copied() const`

  Returns whether the current label was assigned with `copy_label()`.

- `const char *label()` const

  Gets the current label text.

  ```
  void label(const char *text)
  
  Sets the current label pointer.
  ```

  ```
  void label(Fl_Labeltype a, const char *b)
  
  Shortcut to set the label text and type in one call.
  ```

- `Fl_Color labelcolor()` const

  Gets the label color.

- `void labelcolor(Fl_Color c)`

  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
Changes the size of the widget.

- int take_focus ()
  Gives the widget the keyboard focus.

- unsigned int takesevents () const
  Returns if the widget is able to take events.

- int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.

- const char ∗ tooltip () const
  Gets the current tooltip text.

- void tooltip (const char ∗text)
  Sets the current tooltip text.

- Fl_Window ∗ top_window () const
  Returns a pointer to the top-level window for the widget.

- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const
  Gets the widget type.

- void type (uchar t)
  Sets the widget type.

- int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void ∗ user_data () const
  Gets the user data for this widget.

- void user_data (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.

- void user_data (void ∗v)
  Sets the user data for this widget.

- unsigned int visible () const
  Returns whether a widget is visible.

- unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

- void visible_focus (int v)
  Modifies keyboard focus navigation.

- int visible_r () const
  Returns whether a widget and all its parents are visible.

- int w () const
  Gets the widget width.

- Fl_When when () const
  Returns the conditions under which the callback is called.

- void when (uchar i)
  Sets the flags used to decide when a callback is called.

- Fl_Window ∗ window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

- int x () const
  Gets the widget position in its window.

- int y () const
  Gets the widget position in its window.

- virtual ~Fl_Widget ()
  Destroys the widget.
Protected Member Functions

- virtual int alloc_size (int size) const
  Return new size to be allocated for array of fixed size widgets.

- void draw () FL_OVERRIDE
  Draw the widget.

- void init (int t=VERTICAL)

- void on_remove (int) FL_OVERRIDE
  Allow derived groups to act when a child widget is removed from the group.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.

- void draw () FL_OVERRIDE
  Draws the widget.

- void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.

- void draw_children ()
  Draws all children of the group.

- void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.

- virtual int on_insert (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.

- virtual int on_move (int, int)
  Allow derived groups to act when a widget is moved within the group.

- int * sizes ()
  Returns the internal array of widget sizes and positions.

- void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.

- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void draw_box () const
  Draws the widget box according its box style.

- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- void draw_focus () const
  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

- void draw_label () const
  Draws the widget's label at the defined label position.

- void draw_label (int, int, int, int) const
Draws the label in an arbitrary bounding box.

- **Fl_Widget** (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.

- unsigned int flags () const
  Gets the widget flags mask.

- void h (int v)
  Internal use only.

- void set_flag (unsigned int c)
  Sets a flag in the flags mask.

- void w (int v)
  Internal use only.

- void x (int v)
  Internal use only.

- void y (int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from **Fl_Group**

- static Fl_Group ∗ current ()
  Returns the currently active group.

- static void current (Fl_Group ∗g)
  Sets the current group.

Static Public Member Functions inherited from **Fl_Widget**

- static void default_callback (Fl_Widget ∗widget, void ∗data)
  The default callback for all widgets that don’t set a callback.

- static unsigned int label_shortcut (const char ∗t)
  Returns the Unicode value of the ‘&x’ shortcut in a given text.

- static int test_shortcut (const char ∗, const bool require_alt=false)
  Returns true if the given text ∗ contains the entered ‘&x’ shortcut.

Protected Types inherited from **Fl_Widget**

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , Copied_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , Copied.TOOLKIT = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }

  flags possible values enumeration.
12.41.1 Detailed Description

`Fl_Flex` is a container (layout) widget for one row or one column of widgets. It provides flexible positioning of its children either in one row or in one column. `Fl_Flex` is designed to be as simple as possible. You can set individual widget sizes or let `Fl_Flex` position and size the widgets to fit in the container. All "flexible" (i.e. non-fixed size) widgets are assigned the same width or height, respectively. For details see below.

You can set the margins around all children at the inner side of the box frame (if any). `Fl_Flex` supports setting different margin sizes on top, bottom, left, and right sides. The default margin size is 0 on all edges of the container. You can set the gap size between all children. The gap size is always the same between all of its children. This is similar to the 'spacing' of `Fl_Pack`. The default gap size is 0.

`Fl_Flex` can either consist of a single row, i.e. `type(Fl_Flex::HORIZONTAL)` or a single column, i.e. `type(Fl_Flex::VERTICAL)`. The default value is `Fl_Flex::VERTICAL` for consistency with `Fl_Pack` but you can use `type()` to assign a row (`Fl_Flex::HORIZONTAL`) layout. If `type() == Fl_Flex::HORIZONTAL` widgets are resized horizontally to fit in the container and their height is the full `Fl_Flex` height minus border size and margins. You can set a fixed widget width by using `fixed()`.

If `type() == Fl_Flex::VERTICAL` widgets are resized vertically to fit in the container and their width is the full `Fl_Flex` width minus border size and margins. You can set a fixed widget height by using `fixed()`.

To create arbitrary spacing you can use invisible boxes of flexible or fixed sizes (see example below). Alternate constructors let you specify the layout as `Fl_Flex::HORIZONTAL` or `Fl_Flex::VERTICAL` directly. `Fl_Flex::ROW` is an alias of `Fl_Flex::HORIZONTAL` and `Fl_Flex::COLUMN` is an alias of `Fl_Flex::VERTICAL`. The default box type is `FL_NO_BOX` as inherited from `Fl_Group`. You may need to set a box type with a solid background depending on your layout.

**Important:** You should always make sure that the `Fl_Flex` container cannot be resized smaller than its designed minimal size. This can usually be done by setting a `size_range()` on the window as shown in the example below. `Fl_Flex` does not take care of sensible sizes. If it is resized too small the behavior is undefined, i.e. widgets may overlap and/or shrink to zero size.

**Hint:** In many cases `Fl_Flex` can be used as a drop-in replacement for `Fl_Pack`. This is the recommended single row/column container since FLTK 1.4.0. Its resizing behavior is much more predictable (as expected) than that of `Fl_Pack` which "resizes itself to shrink-wrap itself around all of the children". `Fl_Flex` containers can be nested so you can create flexible layouts with multiple columns and rows. However, if your UI design is more complex you may want to use `Fl_Grid` instead.

Example:

```
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Flex.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Button.H>

int main(int argc, char **argv) {
    Fl_Double_Window window(410, 40, "Simple Fl_Flex Demo");
    Fl_Flex flex(5, 5, 400, 30, Fl_Flex::HORIZONTAL);
    Fl_Button b1(0, 0, 0, 0, "File");
    Fl_Button b2(0, 0, 0, 0, "Save");
    Fl_Box bx(0, 0, 0, 0);
    Fl_Button b3(0, 0, 0, 0, "Exit");
    flex.fixed(bx, 60); // set fix width of invisible box
    flex.gap(10);
    flex.end();
    window.resizable(flex);
    window.end();
    window.size_range(300, 30);
    window.show(argc, argv);
    return Fl::run();
}
```

![Figure 12.18 Fl_Flex](image-url)
Since 1.4.0

12.41.2 Member Enumeration Documentation

12.41.2.1 anonymous enum

anonymous enum

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERTICAL</td>
<td>vertical layout (one column)</td>
</tr>
<tr>
<td>HORIZONTAL</td>
<td>horizontal layout (one row)</td>
</tr>
<tr>
<td>COLUMN</td>
<td>alias for VERTICAL</td>
</tr>
<tr>
<td>ROW</td>
<td>alias for HORIZONTAL</td>
</tr>
</tbody>
</table>

12.41.3 Constructor & Destructor Documentation

12.41.3.1 Fl_Flex() [1/4]

Fl_Flex::Fl_Flex (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char ∗ L = 0 )

Construct a new Fl_Flex widget with the given position, size, and label. You can set type(Fl_Flex::HORIZONTAL) or type(Fl_Flex::VERTICAL). The default is type(Fl_Flex::VERTICAL).

Alternate constructors let you specify the layout as Fl_Flex::HORIZONTAL or Fl_Flex::VERTICAL directly. Fl_Flex::ROW is an alias of Fl_Flex::HORIZONTAL and Fl_Flex::COLUMN is an alias of Fl_Flex::VERTICAL.

Parameters

| in  X, Y  | position |
| in  W, H  | size (width and height) |
| in  L     | label (optional) |

See also

    Fl_Flex::Fl_Flex(int direction)
    Fl_Flex::Fl_Flex(int w, int h, int direction)
    Fl_Flex::Fl_Flex(int x, int y, int w, int h, int direction)
    Fl_Flex::Fl_Flex(int x, int y, int w, int h, const char ∗ L)

12.41.3.2 Fl_Flex() [2/4]

Fl_Flex::Fl_Flex (  
  int direction )

Construct a new Fl_Flex widget specifying its layout. Use Fl_Flex::HORIZONTAL (aka Fl_Flex::ROW) or Fl_Flex::VERTICAL (aka Fl_Flex::COLUMN) as the direction argument.

This constructor sets the position and size to (0, 0, 0, 0) which is suitable for nested Fl_Flex widgets. Use one of the other constructors to set the desired position and size as well.
Parameters

\[
\begin{array}{|c|c|}
\hline
\text{in} & \text{direction} \\
\hline
\text{horizontal (row) or vertical (column) layout} \\
\hline
\end{array}
\]

See also

- `Fl_Flex::Fl_Flex(int w, int h, int direction)`
- `Fl_Flex::Fl_Flex(int x, int y, int w, int h, int direction)`
- `Fl_Flex::Fl_Flex(int x, int y, int w, int h, const char *L)`

12.41.3.3 Fl_Flex() [3/4]

\[
\begin{array}{c}
\text{Fl_Flex::Fl_Flex (}
\end{array}
\]

Construct a new Fl_Flex widget specifying its layout and size.

Use Fl_Flex::HORIZONTAL (aka Fl_Flex::ROW) or Fl_Flex::VERTICAL (aka Fl_Flex::COLUMN) as the direction argument.

This constructor sets the position to \((x = 0, y = 0)\) which is suitable for nested Fl_Flex widgets. Use one of the other constructors to set the desired position as well.

Parameters

\[
\begin{array}{|c|c|}
\hline
\text{in} & \text{w,h} \\
\hline
\text{widget size} \\
\hline
\text{in} & \text{direction} \\
\hline
\text{horizontal (row) or vertical (column) layout} \\
\hline
\end{array}
\]

See also

- `Fl_Flex::Fl_Flex(int direction)`
- `Fl_Flex::Fl_Flex(int x, int y, int w, int h, int direction)`
- `Fl_Flex::Fl_Flex(int x, int y, int w, int h, const char *L)`

12.41.3.4 Fl_Flex() [4/4]

\[
\begin{array}{c}
\text{Fl_Flex::Fl_Flex (}
\end{array}
\]

Construct a new Fl_Flex widget specifying its layout, position, and size.

Use Fl_Flex::HORIZONTAL (aka Fl_Flex::ROW) or Fl_Flex::VERTICAL (aka Fl_Flex::COLUMN) as the direction argument.

This constructor sets the position and size of the widget which is suitable for top level Fl_Flex widgets but does not set a widget label. Use Fl_Widget::label() to set one if desired.

Parameters

\[
\begin{array}{|c|c|}
\hline
\text{in} & \text{x,y} \\
\hline
\text{widget position} \\
\hline
\text{in} & \text{w,h} \\
\hline
\text{widget size} \\
\hline
\text{in} & \text{direction} \\
\hline
\text{horizontal (row) or vertical (column) layout} \\
\hline
\end{array}
\]
See also

- `Fl_Flex::Fl_Flex(int direction)`
- `Fl_Flex::Fl_Flex(int w, int h, int direction)`
- `Fl_Flex::Fl_Flex(int x, int y, int w, int h, const char *L)`

### 12.41.4 Member Function Documentation

#### 12.41.4.1 alloc_size()

```cpp
ing int Fl_Flex::alloc_size (int size) const [protected], [virtual]
```

Return new size to be allocated for array of fixed size widgets.
This method is called when the array of fixed size widgets needs to be expanded. The current size is provided (size can be 0). The default method adds 8 to the current size.
This can be used in derived classes to change the allocation strategy. Note that this method only queries the new size which shall be allocated but does not allocate the memory.

**Parameters**

- `in size` current size

**Returns**

- `int new size (to be allocated)`

#### 12.41.4.2 draw()

```cpp
void Fl_Flex::draw (void) [protected], [virtual]
```

Draw the widget.
This will finally calculate the layout of the widget and of all its children if necessary and draw the widget.
Some changes of included children may require a new layout to be calculated. If this is the case the user may need to call `layout()` to make sure everything is calculated properly.

See also

- `layout()`

Implements `Fl_Widget`.

#### 12.41.4.3 end()

```cpp
void Fl_Flex::end ( ) [virtual]
```

Ends automatic child addition and resizes all children.
This marks the `Fl_Flex` widget as changed (need_layout(1)) which forces the widget to calculate its layout depending on all children and whether they have been assigned fix sizes or not right before it is drawn.

See also

- `need_layout(int)`
- `draw()`

#### 12.41.4.4 fixed() [1/3]

```cpp
void Fl_Flex::fixed ( Fl_Widget & w, int size ) [inline]
```

Set the horizontal or vertical size of a child widget.
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>w</th>
<th>widget to be affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>width (Fl_Flex::HORIZONTAL) or height (Fl_Flex::VERTICAL)</td>
</tr>
</tbody>
</table>

See also

fixed(Fl_Widget ∗w, int size)

#### 12.41.4.5 fixed() [2/3]

```cpp
int Fl_Flex::fixed (Fl_Widget ∗w) const
```

Return whether the given widget has a fixed size or resizes dynamically.

**Parameters**

| in | w | widget |

**Returns**

- whether the widget has a fixed size

**Return values**

| 1   | the widget has a fixed size |
| 0   | the widget resizes dynamically |

#### 12.41.4.6 fixed() [3/3]

```cpp
void Fl_Flex::fixed (Fl_Widget ∗child, int size)
```

Set the horizontal or vertical size of a child widget.

This sets either the width or height of a child widget, depending on the type() of the Fl_Flex container (Fl_Flex::HORIZONTAL or Fl_Flex::VERTICAL). The other dimension is set to the full width or height of the Fl_Flex widget minus border and margin sizes. This can be used to set a fixed widget width or height of children of Fl_Flex so they are not resized dynamically. If size is 0 (zero) or negative the widget size is reset to flexible size.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>child</th>
<th>widget to be affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>width (Fl_Flex::HORIZONTAL) or height (Fl_Flex::VERTICAL)</td>
</tr>
</tbody>
</table>

#### 12.41.4.7 gap() [1/2]

```cpp
int Fl_Flex::gap () const [inline]
```

Return the gap size of the widget.

**Returns**

- gap size between all child widgets.
12.41.4.8 gap() [2/2]

```c
void Fl_Flex::gap ( int g ) [inline]
```

Set the gap size of the widget. The gap size is some free space between child widgets. The size must be \(\geq 0\). Negative values are clamped to 0.

**Parameters**

| in | g | gap size |

12.41.4.9 horizontal()

```c
int Fl_Flex::horizontal ( ) const [inline]
```

Returns non-zero (true) if Fl_Flex alignment is horizontal (row mode).

**Returns**

non-zero if Fl_Flex alignment is horizontal

**Return values**

| 1 | if type() == Fl_Flex::HORIZONTAL |
| 0 | if type() == Fl_Flex::VERTICAL |

See class Fl_Flex documentation for details.

12.41.4.10 layout()

```c
void Fl_Flex::layout ( )
```

Calculates the layout of the widget and redraws it. If you change widgets in the Fl_Flex container you should call this method to force recalculation of child widget sizes and positions. This can be useful (necessary) if you hide(), show(), add() or remove() children. Call this method if you need to recalculate widget positions for usage in an algorithm that places widgets at certain positions or when you need to display (show) or hide one or more children depending on the current layout (for instance a side bar). This method also calls redraw() on the Fl_Flex widget.

12.41.4.11 margin() [1/4]

```c
int Fl_Flex::margin ( ) const [inline]
```

Returns the left margin size of the widget. This returns the left margin of the widget which is not necessarily the same as all other margins.

**Note**

This method is useful if you never set different margin sizes.

**See also**

int margin(int *left, int *top, int *right, int *bottom) to get all four margin values.

**Returns**

size of left margin.
12.41.12 margin() [2/4]

int Fl_Flex::margin (  
    int * left,  
    int * top,  
    int * right,  
    int * bottom ) const [inline]

Returns all (four) margin sizes of the widget.  
All margin sizes are returned in the given arguments. If any argument is NULL the respective value is not returned.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>left</th>
<th>returns left margin if not NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>top</td>
<td>returns top margin if not NULL</td>
</tr>
<tr>
<td>in</td>
<td>right</td>
<td>returns right margin if not NULL</td>
</tr>
<tr>
<td>in</td>
<td>bottom</td>
<td>returns bottom margin if not NULL</td>
</tr>
</tbody>
</table>

Returns

whether all margins are equal

Return values

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>all margins have the same size</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>at least one margin has a different size</td>
<td></td>
</tr>
</tbody>
</table>

12.41.13 margin() [3/4]

void Fl_Flex::margin (  
    int left,  
    int top,  
    int right,  
    int bottom ) [inline]

Set the margin sizes at all four edges of the Fl_Flex widget.  
The margin is the free space inside the widget border around all child widgets.  
You must use all four parameters of this method to set the four margins in the order left, top, right, bottom.  
Negative values are set to 0 (zero).  
To set all margins to equal sizes, use margin(int m) which sets all four margins to the same size.

Parameters

| in | left,top,right,bottom | margin sizes, must be >= 0 |

See also

margin(int, int)

12.41.14 margin() [4/4]

void Fl_Flex::margin (  
    int m,  
    int g = -1 ) [inline]

Set the margin and optionally the gap size of the widget.  
This method can be used to set both the margin and the gap size.  
If you don't use the second parameter g or supply a negative value the gap size is not changed.
The margin is the free space inside the widget border around all child widgets. This method sets the margin to the same size at all four edges of the Fl_Flex widget. The gap size $g$ is the free space between child widgets. Negative values do not change the gap value. This is the default if this argument is omitted.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>Margin size, must be $\geq 0$</td>
</tr>
<tr>
<td>$g$</td>
<td>Gap size (ignored, if negative)</td>
</tr>
</tbody>
</table>

See also

**gap(int)**

#### 12.41.4.15 need_layout() [1/2]

```cpp
bool Fl_Flex::need_layout () const [inline]
```

Returns whether layout calculation is required. This should rarely be needed by user code. Used internally in draw().

#### 12.41.4.16 need_layout() [2/2]

```cpp
void Fl_Flex::need_layout ( int set ) [inline]
```

Set or reset the request to calculate the layout of children. This is intended for internal use but can also be used by user code to request layout calculation before the widget is drawn. Call this if you changed attributes or sizes of children to ensure that the layout is calculated properly. Changing other Fl_Flex attributes or resizing the widget does this automatically.

**Note**

Never call this with `$set == 0$` because this would defeat its purpose to recalculate the layout before the widget is drawn.

#### 12.41.4.17 on_remove()

```cpp
void Fl_Flex::on_remove ( int index ) [protected], [virtual]
```

Allow derived groups to act when a child widget is removed from the group. Widgets derived from Fl_Group may store additional data for their children. Overriding this method will allow derived classes to remove these data structures just before the child is removed.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Remove the child at this position in the array_</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Group.

#### 12.41.4.18 resize()

```cpp
void Fl_Flex::resize ( int x, int y, int w, int h ) [virtual]
```
Resize the container and calculate all child positions and sizes.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x, y</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>w, h</td>
<td>width and height</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Widget.

12.41.19 spacing() [1/2]

int Fl_Flex::spacing () const [inline]

Gets the number of extra pixels of blank space that are added between the children.

This method is the same as 'int gap()' and is defined to enable using Fl_Flex as a drop-in replacement of Fl_Pack.

See also

int gap()

12.41.20 spacing() [2/2]

void Fl_Flex::spacing (int i) [inline]

Sets the number of extra pixels of blank space that are added between the children.

This method is the same as 'gap(int)' and is defined to enable using Fl_Flex as a drop-in replacement of Fl_Pack.

See also

void gap(int)

The documentation for this class was generated from the following files:

- Fl_Flex.H
- Fl_Flex.cxx

12.42 Fl_Float_Input Class Reference

The Fl_Float_Input class is a subclass of Fl_Input that only allows the user to type floating point numbers (sign, digits, decimal point, more digits, 'E' or 'e', sign, digits).

#include <Fl_Float_Input.H>

Inheritance diagram for Fl_Float_Input:

```
Fl_Widget
  ↓
Fl_Input_
  ↓
Fl_Input
  ↓
Fl_Float_Input
```

Public Member Functions

- Fl_Float_Input (int X, int Y, int W, int H, const char *l=0)
  
  Creates a new Fl_Float_Input widget using the given position, size, and label string.
Public Member Functions inherited from Fl_Input

• Fl_Input (int, int, int, int, const char *=0)
  Creates a new Fl_Input widget using the given position, size, and label string.
• int handle (int) FL_OVERRIDE
  Handles the specified event.

Public Member Functions inherited from Fl_Input_

• int append (const char *, int l=0, char keep_selection=0)
  Append text at the end.
• bool can_redo () const
  Check if there is a redo action available.
• bool can_undo () const
  Check if the last operation can be undone.
• int copy (int clipboard)
  Put the current selection into the clipboard.
• int copy_cuts ()
  Copies the yank buffer to the clipboard.
• Fl_Color cursor_color () const
  Gets the color of the cursor.
• void cursor_color (Fl_Color n)
  Sets the color of the cursor.
• int cut ()
  Deletes the current selection.
• int cut (int a, int b)
  Deletes all characters between index a and b.
• int cut (int n)
  Deletes the next n bytes rounded to characters before or after the cursor.
• double dvalue () const
  Returns the widget text interpreted as a floating point number.
• Fl_Input_ (int, int, int, int, const char *=0)
  Creates a new Fl_Input_ widget.
• unsigned int index (int i) const
  Returns the character at index i.
• int input_type () const
  Gets the input field type.
• void input_type (int t)
  Sets the input field type.
• int insert (const char *, int l=0)
  Inserts text at the cursor position.
• int insert_position () const
  Gets the position of the text cursor.
• int insert_position (int p)
  Sets the cursor position and mark.
• int insert_position (int p, int m)
  Sets the index for the cursor and mark.
• int ivalue () const
  Returns the widget text interpreted as a signed integer.
• int mark () const
  Gets the current selection mark.
• int mark (int m)
Sets the current selection mark.

- int `maximum_size ()` const
  
  Gets the maximum length of the input field in characters.

- void `maximum_size (int m)`
  
  Sets the maximum length of the input field in characters.

- int `position ()` const
- int `position (int p)`
- int `position (int p, int m)`
- int `readonly ()` const
  
  Gets the read-only state of the input field.

- void `readonly (int b)`
  
  Sets the read-only state of the input field.

- int `redo ()`
  
  Redo previous undo operation.

- int `replace (int b, int e, const char ∗text, int ilen=0)`
  
  Deletes text from `b` to `e` and inserts the new string `text`.

- void `resize (int, int, int, int)` FL_OVERRIDE
  
  Changes the size of the widget.

- int `shortcut ()` const
  
  Return the shortcut key associated with this widget.

- void `shortcut (int s)`
  
  Sets the shortcut key associated with this widget.

- int `size ()` const
  
  Returns the number of bytes in `value()`.

- void `size (int W, int H)`
  
  Sets the width and height of this widget.

- int `static_value (const char ∗)`
  
  Changes the widget text.

- int `static_value (const char ∗, int)`
  
  Changes the widget text.

- int `tab_nav ()` const
  
  Gets whether the Tab key causes focus navigation in multiline input fields or not.

- void `tab_nav (int val)`
  
  Sets whether the Tab key does focus navigation, or inserts tab characters into `Fl_Multiline_Input`.

- `Fl_Color textcolor ()` const
  
  Gets the color of the text in the input field.

- void `textcolor (Fl_Color n)`
  
  Sets the color of the text in the input field.

- `FL_Font textfont ()` const
  
  Gets the font of the text in the input field.

- void `textfont (FL_Font s)`
  
  Sets the font of the text in the input field.

- `FL_Fontsize textsize ()` const
  
  Gets the size of the text in the input field.

- void `textsize (FL_Fontsize s)`
  
  Sets the size of the text in the input field.

- int `undo ()`
  
  Undoes previous changes to the text buffer.

- const char ∗`value ()` const
  
  Returns the text displayed in the widget.

- int `value (const char ∗)`
Changes the widget text.

- int value (const char *, int)
  Changes the widget text.
- int value (double value)
  Changes the widget text to a floating point number ("%g").
- int value (int value)
  Changes the widget text to a signed integer number.
- void wrap (int b)
  Sets the word wrapping state of the input field.
- ~Fl_Input_()
  Destroys the widget.

Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen()
- void _set_fullscreen()
- void activate()
  Activates the widget.
- unsigned int active() const
  Returns whether the widget is active.
- int active_r() const
  Returns whether the widget and all of its parents are active.
- Fl_Align align() const
  Gets the label alignment.
- void align(Fl_Align alignment)
  Sets the label alignment.
- long argument() const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument(long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * as_gl_window() const
- virtual Fl_Group * as_group()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * as_group() const
- virtual Fl_Window * as_window()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * as_window() const
- void bind_deimage(Fl_image const * img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage(int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image(Fl_image * img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image(int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box() const
  Gets the box type of the widget.
• **void** box (Fl_Boxtype new_box)
  *Sets the box type for the widget.*

• **Fl_Callback_p** callback () const
  *Gets the current callback function for the widget.*

• **void** callback (Fl_Callback cb)
  *Sets the current callback function for the widget.*

• **void** callback (Fl_Callback cb, Fl_Callback_User_Data *p, bool auto_free)
  *Sets the current callback function and managed user data for the widget.*

• **void** callback (Fl_Callback cb, void *p)
  *Sets the current callback function and data for the widget.*

• **void** callback (Fl_Callback0 cb)
  *Sets the current callback function for the widget.*

• **void** callback (Fl_Callback1 cb, long p=0)
  *Sets the current callback function for the widget.*

• **unsigned int** changed () const
  *Checks if the widget value changed since the last callback.*

• **void** clear_active ()
  *Marks the widget as inactive without sending events or changing focus.*

• **void** clear_changed ()
  *Marks the value of the widget as unchanged.*

• **void** clear_damage (uchar c=0)
  *Clears or sets the damage flags.*

• **void** clear_output ()
  *Sets a widget to accept input.*

• **void** clear_visible ()
  *Hides the widget.*

• **void** clear_visible_focus ()
  *Disables keyboard focus navigation with this widget.*

• **Fl_Color** color () const
  *Gets the background color of the widget.*

• **void** color (Fl_Color bg)
  *Sets the background color of the widget.*

• **void** color (Fl_Color bg, Fl_Color sel)
  *Sets the background and selection color of the widget.*

• **Fl_Color** color2 () const
  *For back compatibility only.*

• **void** color2 (unsigned a)
  *For back compatibility only.*

• **int** contains (const Fl_Widget *w) const
  *Checks if w is a child of this widget.*

• **void** copy_label (const char *new_label)
  *Sets the current label.*

• **void** copy_tooltip (const char *text)
  *Sets the current tooltip text.*

• **uchar** damage () const
  *Returns non-zero if draw() needs to be called.*

• **void** damage (uchar c)
  *Sets the damage bits for the widget.*

• **void** damage (uchar c, int x, int y, int w, int h)
  *Sets the damage bits for an area inside the widget.*

• **int** damage_resize (int, int, int, int)
Internal use only.

- **void** **deactivate** ()
  
  Deactivates the widget.

- **FL_Image** **∗** **deimage** ()
  
  Gets the image that is used as part of the widget label when in the inactive state.

- **const FL_Image** **∗** **deimage** () const
  
  Gets the image that is used as part of the widget label when in the inactive state.

- **void** **deimage** (**FL_Image** &img)
  
  Sets the image to use as part of the widget label when in the inactive state.

- **void** **deimage** (**FL_Image** **∗** img)
  
  Sets the image to use as part of the widget label when in the inactive state.

- **int** **deimage_bound** () const
  
  Returns whether the inactive image is managed by the widget.

- **void** **do_callback** (**FL_Callback_Reason** reason=**FL_REASON_UNKNOWN**)
  
  Calls the widget callback function with default arguments.

- **void** **do_callback** (**FL_Widget** **∗** wgt, **long** arg, **FL_Callback_Reason** reason=**FL_REASON_UNKNOWN**)
  
  Calls the widget callback function with arbitrary arguments.

- **void** **do_callback** (**FL_Widget** **∗** wgt, **void** **∗** arg=0, **FL_Callback_Reason** reason=**FL_REASON_UNKNOWN**)
  
  Calls the widget callback function with arbitrary arguments.

- **void** **draw_label** (int, int, int, int, **FL_Align**) const
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int** **h** () const
  
  Gets the widget height.

- virtual **void** **hide** ()
  
  Makes a widget invisible.

- **FL_Image** **∗** **image** ()
  
  Gets the image that is used as part of the widget label when in the active state.

- **const FL_Image** **∗** **image** () const
  
  Gets the image that is used as part of the widget label when in the active state.

- **void** **image** (**FL_Image** &img)
  
  Sets the image to use as part of the widget label when in the active state.

- **void** **image** (**FL_Image** **∗** img)
  
  Sets the image to use as part of the widget label when in the active state.

- **int** **image_bound** () const
  
  Returns whether the image is managed by the widget.

- **int** **inside** (**const FL_Widget** **∗** wgt) const
  
  Checks if this widget is a child of wgt.

- **int** **is_label_copied** () const
  
  Returns whether the current label was assigned with copy_label().

- **const char** **∗** **label** () const
  
  Gets the current label text.

- **void** **label** (**const char** **∗** text)
  
  Sets the current label pointer.

- **void** **label** (**FL_Labeltype** a, **const char** **∗** b)
  
  Shortcut to set the label text and type in one call.

- **FL_Color** **labelcolor** () const
  
  Gets the label color.

- **void** **labelcolor** (**FL_Color** c)
  
  Sets the label color.

- **FL_Font** **labelfont** () const
  
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group *parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses ' &' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses ' &' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
Gives the widget the keyboard focus.

- `unsigned int takeEvents()` const
  Returns if the widget is able to take events.

- `int testShortcut()` const
  Returns true if the widget’s label contains the entered ‘&x’ shortcut.

- `const char * tooltip()` const
  Gets the current tooltip text.

- `void tooltip(const char *text)`
  Sets the current tooltip text.

- `Fl_Window * top_window()` const
  Returns a pointer to the top-level window for the widget.

- `Fl_Window * top_window_offset(int &xoff, int &yoff)` const
  Finds the x/y offset of the current widget relative to the top-level window.

- `uchar type()` const
  Gets the widget type.

- `void type(uchar t)`
  Sets the widget type.

- `int useAccentsMenu()` const
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- `void *user_data()` const
  Gets the user data for this widget.

- `void user_data(Fl_Callback_User_Data *v, bool auto_free)`
  Sets the user data for this widget.

- `void user_data(void *v)`
  Sets the user data for this widget.

- `unsigned int visible()` const
  Returns whether a widget is visible.

- `unsigned int visible_focus()` const
  Checks whether this widget has a visible focus.

- `void visible_focus(int v)`
  Modifies keyboard focus navigation.

- `int visible_r()` const
  Returns whether a widget and all its parents are visible.

- `int w()` const
  Gets the widget width.

- `Fl_When when()` const
  Returns the conditions under which the callback is called.

- `void when(uchar i)`
  Sets the flags used to decide when a callback is called.

- `Fl_Window * window()` const
  Returns a pointer to the nearest parent window up the widget hierarchy.

- `int x()` const
  Gets the widget position in its window.

- `int y()` const
  Gets the widget position in its window.

- `virtual ~Fl_Widget()`
  Destroys the widget.
Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget ∗widget, void ∗data)
  The default callback for all widgets that don't set a callback.
• static unsigned int label_shortcut (const char ∗t)
  Returns the Unicode value of the '&x' shortcut in a given text.
• static int test_shortcut (const char ∗t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Static Public Attributes inherited from Fl_Input

• static const char ∗copy_menu_text = "Copy"
  [this text may be customized at run-time]
• static const char ∗cut_menu_text = "Cut"
  [this text may be customized at run-time]
• static const char ∗paste_menu_text = "Paste"
  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPYED_TOOLIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19, NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
flags possible values enumeration.

Protected Member Functions inherited from Fl_Input

• void draw () FL_OVERRIDE
  Draws the widget.
• int handle_key ()
  Handles a keystroke.
• int handle_rmb ()
  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

• int apply_undo ()
  Apply the current undo/redo operation.
• void drawtext (int, int, int, int)
  Draws the text in the passed bounding box.
• void drawtext (int, int, int, int, bool draw_active)
  Draws the text in the passed bounding box.
• void handle_mouse (int, int, int, int keepmark=0)
Handles mouse clicks and mouse moves.

- `int handle_text(int e, int, int, int, int)`
  Handles all kinds of text field related events.

- `int line_end(int i) const`
  Finds the end of a line.

- `int line_start(int i) const`
  Finds the start of a line.

- `int linesPerPage()`

- `void maybe_do_callback(Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`

- `int up_down_position(int, int keepmark=0)`
  Moves the cursor to the column given by `up_down_pos`.

- `int word_end(int i) const`
  Finds the end of a word.

- `int word_start(int i) const`
  Finds the start of a word.

- `int xscroll()`
  Gets the page length.

- `int yscroll()`
  Gets the vertical scroll position.

- `void yscroll(int yOffset)`

Protected Member Functions inherited from `Fl_Widget`

- `void clear_flag(unsigned int c)`
  Clears a flag in the flags mask.

- `void draw_backdrop()`
  If `FL_ALIGN_IMAGE_BACKDROP` is set, the image or deimage will be drawn.

- `void draw_box()`
  Draws the widget box according to its box style.

- `void draw_box(Fl_Boxtype t, Fl_Color c)`
  Draws a box of type `t`, of color `c` at the widget’s position and size.

- `void draw_box(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c)`
  Draws a box of type `t`, of color `c` at the position `X, Y` and size `W,H`.

- `void draw_focus()`
  Draws a focus rectangle around the widget.

- `void draw_focus(Fl_Boxtype t, int X, int Y, int W, int H)`
  Draws a focus rectangle around the widget.

- `void draw_focus(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg)`
  Draws a focus box for the widget at the given position and size.

- `void draw_label()`
  Draws the widget’s label at the defined label position.

- `void draw_label(int, int, int, int)`
  Draws the label in an arbitrary bounding box.

- `Fl_Widget(int x, int y, int w, int h, const char *label=0L)`
  Creates a widget at the given position and size.

- `unsigned int flags()`
  Gets the widget flags mask.

- `void h(int v)`
  Internal use only.

- `void set_flag(unsigned int c)`
  Sets a flag in the flags mask.

- `void w(int v)`
  Internal use only.
12.42.1 Detailed Description

The Fl_Float_Input class is a subclass of Fl_Input that only allows the user to type floating point numbers (sign, digits, decimal point, more digits, ‘E’ or ‘e’, sign, digits).

12.42.2 Constructor & Destructor Documentation

12.42.2.1 Fl_Float_Input()

Fl_Float_Input::Fl_Float_Input (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char ∗ l = 0 )

Creates a new Fl_Float_Input widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX.

Inherited destructor destroys the widget and any value associated with it.

The documentation for this class was generated from the following files:

- Fl_Float_Input.H
- Fl_Input.cxx

12.43 Fl_FormsBitmap Class Reference

Forms compatibility Bitmap Image Widget.

#include <Fl_FormsBitmap.H>

Inheritance diagram for Fl_FormsBitmap:

```
Fl_Widget
    └── Fl_FormsBitmap
```

Public Member Functions

- Fl_Bitmap ∗ bitmap () const  
  
  Gets a the current associated Fl_Bitmap objects.
- void bitmap (Fl_Bitmap ∗B)
  
  Sets a new bitmap.
- Fl_FormsBitmap (Fl_Boxtype, int, int, int, int, const char ∗=0)
  
  Creates a bitmap widget from a box type, position, size and optional label specification.
- void set (int W, int H, const uchar ∗=bits)
  
  Sets a new bitmap bits with size W,H.

Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
Activates the widget.
• unsigned int active() const
  Returns whether the widget is active.
• int active_r() const
  Returns whether the widget and all of its parents are active.
• Fl_Align align() const
  Gets the label alignment.
• void align(Fl_Align alignment)
  Sets the label alignment.
• long argument() const
  Gets the current user data (long) argument that is passed to the callback function.
• void argument(long v)
  Sets the current user data (long) argument that is passed to the callback function.
• virtual class Fl_Gl_Window * as_gl_window()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const * as_gl_window() const
• virtual Fl_Group * as_group()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• virtual Fl_Group const * as_group() const
• virtual Fl_Window * as_window()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const * as_window() const
• void bind_deimage(Fl_Image * img)
  Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage(int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image(Fl_Image * img)
  Sets the image to use as part of the widget label when in the active state.
• void bind_image(int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• Fl_Boxtype box() const
  Gets the box type of the widget.
• void box(Fl_Boxtype new_box)
  Sets the box type for the widget.
• Fl_Callback_p callback() const
  Gets the current callback function for the widget.
• void callback(Fl_Callback * cb)
  Sets the current callback function for the widget.
• void callback(Fl_Callback * cb, Fl_Callback_User_Data * p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback(Fl_Callback * cb, void * p)
  Sets the current callback function and data for the widget.
• void callback(Fl_Callback0 * cb)
  Sets the current callback function for the widget.
• void callback(Fl_Callback1 * cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed() const
  Checks if the widget value changed since the last callback.
• void clear_active()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed()
Marks the value of the widget as unchanged.

• void clear_damage (uchar c=0)
  Clears or sets the damage flags.

• void clear_output ()
  Sets a widget to accept input.

• void clear_visible ()
  Hides the widget.

• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

• Fl_Color color () const
  Gets the background color of the widget.

• void color (Fl_Color bg)
  Sets the background color of the widget.

• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

• Fl_Color color2 () const
  For back compatibility only.

• void color2 (unsigned a)
  For back compatibility only.

• int contains (const Fl_Widget ∗w) const
  Checks if w is a child of this widget.

• void copy_label (const char ∗new_label)
  Sets the current label.

• void copy_tooltip (const char ∗text)
  Sets the current tooltip text.

• uchar damage () const
  Returns non-zero if draw() needs to be called.

• void damage (uchar c)
  Sets the damage bits for the widget.

• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

• int damage_resize (int, int, int, int)
  Internal use only.

• void deactivate ()
  Deactivates the widget.

• Fl_Image ∗deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

• const Fl_Image ∗deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

• void deimage (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the inactive state.

• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

• void do_callback (Fl_Widget ∗widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void do_callback (Fl_Widget ∗widget, void ∗arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• int h () const
  
  Gets the widget height.

• virtual int handle (int event)
  
  Handles the specified event.

• virtual void hide ()
  
  Makes a widget invisible.

• Fl_Image ∗ image ()
  
  Gets the image that is used as part of the widget label when in the active state.

• const Fl_Image ∗ image () const
  
  Gets the image that is used as part of the widget label when in the active state.

• void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the active state.

• void image (Fl_Image ∗img)
  
  Sets the image to use as part of the widget label when in the active state.

• int image_bound () const
  
  Returns whether the image is managed by the widget.

• int inside (const Fl_Widget ∗wgt) const
  
  Checks if this widget is a child of wgt.

• int is_label_copied () const
  
  Returns whether the current label was assigned with copy_label().

• const char ∗ label () const
  
  Gets the current label text.

• void label (const char ∗text)
  
  Sets the current label pointer.

• void label (Fl_Labeltype a, const char ∗b)
  
  Shortcut to set the label text and type in one call.

• Fl_Color labelcolor () const
  
  Gets the label color.

• void labelcolor (Fl_Color c)
  
  Sets the label color.

• Fl_Font labelfont () const
  
  Gets the font to use.

• void labelfont (Fl_Font f)
  
  Sets the font to use.

• Fl_Fontsize labelsize () const
  
  Gets the font size in pixels.

• void labelsize (Fl_Fontsize pix)
  
  Sets the font size in pixels.

• Fl_Labeltype labeltype () const
  
  Gets the label type.

• void labeltype (Fl_Labeltype a)
  
  Sets the label type.

• void measure_label (int &ww, int &hh) const
  
  Sets width ww and height hh accordingly with the label size.

• bool needs_keyboard () const
  
  Returns whether this widget needs a keyboard.

• void needs_keyboard (bool needs)
  
  Sets whether this widget needs a keyboard.

• unsigned int output () const
• Fl_Group * parent () const
  Returns a pointer to the parent widget.

• void parent (Fl_Group * p)
  Internal use only - "for hacks only".

• void position (int X, int Y)
  Repositions the window or widget.

• void redraw ()
  Schedules the drawing of the widget.

• void redraw_label ()
  Schedules the drawing of the label.

• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.

• Fl_Color selection_color () const
  Gets the selection color.

• void selection_color (Fl_Color a)
  Sets the selection color.

• void set_active ()
  Marks the widget as active without sending events or changing focus.

• void set_changed ()
  Marks the value of the widget as changed.

• void set_output ()
  Sets a widget to output only.

• void set_visible ()
  Makes the widget visible.

• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.

• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.

• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

• virtual void show ()
  Makes a widget visible.

• void size (int W, int H)
  Changes the size of the widget.

• int take_focus ()
  Gives the widget the keyboard focus.

• unsigned int takesevents () const
  Returns if the widget is able to take events.

• int testShortcut ()
  Returns true if the widget's label contains the entered 'x' shortcut.

• const char * tooltip () const
  Gets the current tooltip text.

• void tooltip (const char *text)
  Sets the current tooltip text.

• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.

• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void * user_data () const
  Gets the user data for this widget.

• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

• void user_data (void *v)
  Sets the user data for this widget.

• unsigned int visible () const
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

• void visible_focus (int v)
  Modifies keyboard focus navigation.

• int visible_r () const
  Returns whether a widget and all its parents are visible.

• int w () const
  Gets the widget width.

• Fl_When when () const
  Returns the conditions under which the callback is called.

• void when (uchar i)
  Sets the flags used to decide when a callback is called.

• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  Gets the widget position in its window.

• int y () const
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  Destroys the widget.

Protected Member Functions

• void draw () FL_OVERRIDE
  Draws the bitmap and its associated box.

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.

• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• void draw_box () const
  Draws the widget box according its box style.

• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

• void draw_focus () const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
• void draw_label () const
  Draws the widget’s label at the defined label position.
• void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
• Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.
• unsigned int flags () const
  Gets the widget flags mask.
• void h (int v)
  Internal use only.
• void set_flag (unsigned int c)
  Sets a flag in the flags mask.
• void w (int v)
  Internal use only.
• void x (int v)
  Internal use only.
• void y (int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget ∗widget, void ∗data)
  The default callback for all widgets that don’t set a callback.
• static unsigned int label_shortcut (const char ∗t)
  Returns the Unicode value of the ‘&x’ shortcut in a given text.
• static int test_shortcut (const char ∗t, const bool require_alt=false)
  Returns true if the given text t contains the entered ‘&x’ shortcut.

Protected Types inherited from Fl_Widget

• enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

12.43.1 Detailed Description

Forms compatibility Bitmap Image Widget.
12.43.2 Member Function Documentation

12.43.2.1 draw()

```cpp
void Fl_FormsBitmap::draw()
    void ) [protected], [virtual]
```

Draws the bitmap and its associated box.
Implements Fl_Widget.

12.43.2.2 set()

```cpp
void Fl_FormsBitmap::set(int W, int H, const uchar ∗ bits)
```

Sets a new bitmap bits with size W,H.
Deletes the previous one.
The documentation for this class was generated from the following files:

- Fl_FormsBitmap.H
- forms_bitmap.cxx

12.44 Fl_FormsPixmap Class Reference

Forms pixmap drawing routines.
#include <Fl_FormsPixmap.H>

Inheritance diagram for Fl_FormsPixmap:

```
Fl_Widget
    Fl_FormsPixmap
```

Public Member Functions

- `Fl_FormsPixmap (Fl_Boxtype t, int X, int Y, int W, int H, const char ∗L=0)`
  Creates a new Fl_FormsPixmap widget using the given box type, position, size and label string.

- `Fl_Pixmap ∗ Pixmap () const`
  Get the internal pixmap pointer.

- `void Pixmap (Fl_Pixmap ∗ B)`
  Set the internal pixmap pointer to an existing pixmap.

- `void set (char ∗ const ∗ bits)`
  Set/create the internal pixmap using raw data.

Public Member Functions inherited from Fl_Widget

- `void _clear_fullscreen ()`
- `void _set_fullscreen ()`
- `void activate ()`
  Activates the widget.

- `unsigned int active () const`
  Returns whether the widget is active.

- `int active_r () const`
  Returns whether the widget and all of its parents are active.

- `Fl_Align align () const`
  Returns whether the widget and all of its parents are active.
Gets the label alignment.

- void **align (Fl_Align alignment)**
  Sets the label alignment.

- long **argument () const**
  Gets the current user data (long) argument that is passed to the callback function.

- void **argument (long v)**
  Sets the current user data (long) argument that is passed to the callback function.

- virtual class Fl_Gl_Window ∗ **as_gl_window ()**
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const ∗ **as_gl_window () const**

- virtual Fl_Group ∗ **as_group ()**
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- virtual Fl_Group const ∗ **as_group () const**

- virtual Fl_Window ∗ **as_window ()**
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const ∗ **as_window () const**

- void **bind_deimage (Fl_Image ∗ img)**
  Sets the image to use as part of the widget label when in the inactive state.

- void **bind_deimage (int f)**
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void **bind_image (Fl_Image ∗ img)**
  Sets the image to use as part of the widget label when in the active state.

- void **bind_image (int f)**
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- Fl_Boxtype **box () const**
  Gets the box type of the widget.

- void **box (Fl_Boxtype new_box)**
  Sets the box type for the widget.

- Fl_Callback_p **callback () const**
  Gets the current callback function for the widget.

- void **callback (Fl_Callback ∗ cb)**
  Sets the current callback function for the widget.

- void **callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)**
  Sets the current callback function and managed user data for the widget.

- void **callback (Fl_Callback ∗ cb, void ∗ p)**
  Sets the current callback function and data for the widget.

- void **callback (Fl_Callback0 ∗ cb)**
  Sets the current callback function for the widget.

- void **callback (Fl_Callback1 ∗ cb, long p=0)**
  Sets the current callback function for the widget.

- unsigned int **changed () const**
  Checks if the widget value changed since the last callback.

- void **clear_active ()**
  Marks the widget as inactive without sending events or changing focus.

- void **clear_changed ()**
  Marks the value of the widget as unchanged.

- void **clear_damage (uchar c=0)**
  Clears or sets the damage flags.

- void **clear_output ()**
  Sets a widget to accept input.

- void **clear_visible ()**
Hides the widget.

- void clear_visible_focus ()
  
  Disables keyboard focus navigation with this widget.

- Fl_Color color () const
  
  Gets the background color of the widget.

- void color (Fl_Color bg)
  
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  
  Sets the background and selection color of the widget.

- Fl_Color color2 () const
  
  For back compatibility only.

- void color2 (unsigned a)
  
  For back compatibility only.

- int contains (const Fl_Widget ∗w) const
  
  Checks if w is a child of this widget.

- void copy_label (const char ∗new_label)
  
  Sets the current label.

- void copy_tooltip (const char ∗text)
  
  Sets the current tooltip text.

- uchar damage () const
  
  Returns non-zero if draw() needs to be called.

- void damage (uchar c)
  
  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)
  
  Sets the damage bits for an area inside the widget.

- int damage_resize (int, int, int, int)
  
  Internal use only.

- void deactivate ()
  
  Deactivates the widget.

- Fl_Image ∗deimage ()
  
  Gets the image that is used as part of the widget label when in the inactive state.

- const Fl_Image ∗deimage () const
  
  Gets the image that is used as part of the widget label when in the inactive state.

- void deimage (Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the inactive state.

- void deimage (Fl_Image ∗img)
  
  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const
  
  Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with default arguments.

- void do_callback (Fl_Widget ∗widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with arbitrary arguments.

- void do_callback (Fl_Widget ∗widget, void ∗arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with arbitrary arguments.

- void draw_label (int, int, int, int, Fl_Align) const
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- int h () const
  
  Gets the widget height.

- virtual int handle (int event)
  
  Handles the specified event.
• virtual void hide ()
  
  Makes a widget invisible.

• Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.

• const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.

• void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.

• void image (Fl_Image * img)

  Sets the image to use as part of the widget label when in the active state.

• int image_bound () const

  Returns whether the image is managed by the widget.

• int inside (const Fl_Widget * wgt) const

  Checks if this widget is a child of wgt.

• int is_label_copied () const

  Returns whether the current label was assigned with copy_label().

• const char * label () const

  Gets the current label text.

• void label (const char * text)

  Sets the current label pointer.

• void label (Fl_Labeltype a, const char * b)

  Shortcut to set the label text and type in one call.

• Fl_Color labelcolor () const

  Gets the label color.

• void labelcolor (Fl_Color c)

  Sets the label color.

• Fl_Font labelfont () const

  Gets the font to use.

• void labelfont (Fl_Font f)

  Sets the font to use.

• Fl_Fontsize labelsize () const

  Gets the font size in pixels.

• void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.

• Fl_Labeltype labeltype () const

  Gets the label type.

• void labeltype (Fl_Labeltype a)

  Sets the label type.

• void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.

• bool needs_keyboard () const

  Returns whether this widget needs a keyboard.

• void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.

• unsigned int output () const

  Returns if a widget is used for output only.

• Fl_Group * parent () const

  Returns a pointer to the parent widget.

• void parent (Fl_Group * p)

  Internal use only - “for hacks only”.

• void position (int X, int Y)
Repositions the window or widget.

- **void redraw ()**
  Schedules the drawing of the widget.

- **void redraw_label ()**
  Schedules the drawing of the label.

- **virtual void resize (int x, int y, int w, int h)**
  Changes the size or position of the widget.

- **Fl_Color selection_color () const**
  Gets the selection color.

- **void selection_color (Fl_Color a)**
  Sets the selection color.

- **void set_active ()**
  Marks the widget as active without sending events or changing focus.

- **void set_changed ()**
  Marks the value of the widget as changed.

- **void set_output ()**
  Sets a widget to output only.

- **void set_visible ()**
  Makes the widget visible.

- **void set_visible_focus ()**
  Enables keyboard focus navigation with this widget.

- **int shortcut_label () const**
  Returns whether the widget's label uses '&' to indicate shortcuts.

- **void shortcut_label (int value)**
  Sets whether the widget's label uses '&' to indicate shortcuts.

- **virtual void show ()**
  Makes a widget visible.

- **void size (int W, int H)**
  Changes the size of the widget.

- **int take_focus ()**
  Gives the widget the keyboard focus.

- **unsigned int takesevents () const**
  Returns if the widget is able to take events.

- **int test_shortcut ()**
  Returns true if the widget's label contains the entered '&x' shortcut.

- **const char * tooltip () const**
  Gets the current tooltip text.

- **void tooltip (const char *text)**
  Sets the current tooltip text.

- **Fl_Window * top_window () const**
  Returns a pointer to the top-level window for the widget.

- **Fl_Window * top_window_offset (int &xoff, int &yoff) const**
  Finds the x/y offset of the current widget relative to the top-level window.

- **uchar type () const**
  Gets the widget type.

- **void type (uchar t)**
  Sets the widget type.

- **int use_accents_menu ()**
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- **void * user_data () const**
  Gets the user data for this widget.
• void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)
  
  *Sets the user data for this widget.*

• void **user_data** (void ∗v)
  
  *Sets the user data for this widget.*

• unsigned int **visible** () const
  
  *Returns whether a widget is visible.*

• unsigned int **visible_focus** () const
  
  *Checks whether this widget has a visible focus.*

• void **visible_focus** (int v)
  
  *Modifies keyboard focus navigation.*

• int **visible_r** () const
  
  *Returns whether a widget and all its parents are visible.*

• int **w** () const
  
  *Gets the widget width.*

• Fl_When **when** () const
  
  *Returns the conditions under which the callback is called.*

• void **when** (uchar i)
  
  *Sets the flags used to decide when a callback is called.*

• Fl_Window ∗ **window** () const
  
  *Returns a pointer to the nearest parent window up the widget hierarchy.*

• int **x** () const
  
  *Gets the widget position in its window.*

• int **y** () const
  
  *Gets the widget position in its window.*

• virtual ∼**Fl_Widget** ()
  
  *Destroys the widget.*

### Protected Member Functions

• void **draw** () FL_OVERRIDE
  
  *Draws the widget.*

### Protected Member Functions inherited from **Fl_Widget**

• void **clear_flag** (unsigned int c)
  
  *Clears a flag in the flags mask.*

• void **draw_backdrop** () const
  
  *If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.*

• void **draw_box** () const
  
  *Draws the widget box according its box style.*

• void **draw_box** (Fl_Boxtype t, Fl_Color c) const
  
  *Draws a box of type t, of color c at the widget's position and size.*

• void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  *Draws a box of type t, of color c at the position X,Y and size W,H.*

• void **draw_focus** () const
  
  *Draws a focus rectangle around the widget.*

• void **draw_focus** (Fl_Boxtype t, int X, int Y, int W, int H) const
  
  *Draws a focus rectangle around the widget.*

• void **draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  *Draws a focus box for the widget at the given position and size.*

• void **draw_label** () const
  
  *Draws the widget's label at the defined label position.*
• void **draw_label** (int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.

• **Fl_Widget**(int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.

• unsigned int **flags**( ) const
  
  Gets the widget flags mask.

• void **h**( int v)
  
  Internal use only.

• void **set_flag**( unsigned int c)

  Sets a flag in the flags mask.

• void **w**( int v)
  
  Internal use only.

• void **x**( int v)
  
  Internal use only.

• void **y**( int v)
  
  Internal use only.

**Additional Inherited Members**

**Static Public Member Functions inherited from Fl_Widget**

• static void **default_callback**( Fl_Widget *widget, void *data)
  
  The default callback for all widgets that don’t set a callback.

• static unsigned int **label_shortcut**( const char *t)

  Returns the Unicode value of the ‘&x’ shortcut in a given text.

• static int **test_shortcut**( const char *, const bool require_alt=false)

  Returns true if the given text **t** contains the entered ‘&x’ shortcut.

**Protected Types inherited from Fl_Widget**

• enum {
  
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }

  flags possible values enumeration.

12.44.1 Detailed Description

Forms pixmap drawing routines.

12.44.2 Constructor & Destructor Documentation

12.44.2.1 Fl_FormsPixmap()

Fl_FormsPixmap::Fl_FormsPixmap (  
  Fl_Boxtype t,
  int X,

Generated by Doxygen
int Y,
int W,
int H,
    const char * L = 0)

Creates a new Fl_FormsPixmap widget using the given box type, position, size and label string.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>box type</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y,W,H</td>
<td>position and size</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

### 12.44.3 Member Function Documentation

#### 12.44.3.1 draw()

```c
void Fl_FormsPixmap::draw () [protected, [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```c
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();  // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

#### 12.44.3.2 Pixmap()

```c
void Fl_FormsPixmap::Pixmap ( Fl_Pixmap * B ) [inline]
```

Set the internal pixmap pointer to an existing pixmap.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>B</th>
<th>existing pixmap</th>
</tr>
</thead>
</table>

#### 12.44.3.3 set()

```c
void Fl_FormsPixmap::set ( char *const * bits )
```

Set/create the internal pixmap using raw data.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>bits</th>
<th>raw data</th>
</tr>
</thead>
</table>

The documentation for this class was generated from the following files:

- Fl_FormsPixmap.H
- forms_pixmap.cxx

### 12.45 Fl_FormsText Class Reference

Inheritance diagram for Fl_FormsText:
Public Member Functions

- **Fl_FormsText** (Fl_Boxtype b, int X, int Y, int W, int H, const char *l=0)

Public Member Functions inherited from **Fl_Widget**

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  
  Activates the widget.
- unsigned int active () const
  
  Returns whether the widget is active.
- int active_r () const
  
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  
  Gets the label alignment.
- void align (Fl_Align alignment)
  
  Sets the label alignment.
- long argument () const
  
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * as_gl_window () const
- virtual Fl_Group * as_group ()
  
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * as_group () const
- virtual Fl_Window * as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * as_window () const
- void bind_deimage (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  
  Sets the box type for the widget.
- Fl_Callback * callback () const
  
  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)
Sets the current callback function for the widget.

- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.

- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.

- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.

- unsigned int changed () const

  Checks if the widget value changed since the last callback.

- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()

  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)

  Clears or sets the damage flags.

- void clear_output ()

  Sets a widget to accept input.

- void clear_visible ()

  Hides the widget.

- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.

- Fl_Color color () const

  Gets the background color of the widget.

- void color (Fl_Color bg)

  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.

- Fl_Color color2 () const

  For back compatibility only.

- void color2 (unsigned a)

  For back compatibility only.

- int contains (const Fl_Widget ∗w) const

  Checks if w is a child of this widget.

- void copy_label (const char ∗new_label)

  Sets the current label.

- void copy_tooltip (const char ∗text)

  Sets the current tooltip text.

- uchar damage () const

  Returns non-zero if draw() needs to be called.

- void damage (uchar c)

  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.

- int damage_resize (int, int, int)

  Internal use only.

- void deactivate ()

  Deactivates the widget.

- Fl_Image ∗deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image * deimage () const
  
  Gets the image that is used as part of the widget label when in the inactive state.

• void deimage (Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the inactive state.

• void deimage (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the inactive state.

• int deimage_bound () const
  
  Returns whether the inactive image is managed by the widget.

• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with default arguments.

• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with arbitrary arguments.

• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with arbitrary arguments.

• void draw_label (int, int, int, int, Fl_Align) const
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• int h () const
  
  Gets the widget height.

• virtual int handle (int event)
  
  Handles the specified event.

• virtual void hide ()
  
  Makes a widget invisible.

• Fl_Image * image ()
  
  Gets the image that is used as part of the widget label when in the active state.

• const Fl_Image * image () const
  
  Gets the image that is used as part of the widget label when in the active state.

• void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the active state.

• void image (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the active state.

• int image_bound () const
  
  Returns whether the image is managed by the widget.

• int inside (const Fl_Widget *wgt) const
  
  Checks if this widget is a child of wgt.

• int is_label_copied () const
  
  Returns whether the current label was assigned with copy_label().

• const char * label () const
  
  Gets the current label text.

• void label (const char *text)
  
  Sets the current label pointer.

• void label (Fl_Labeltype a, const char *b)
  
  Shortcut to set the label text and type in one call.

• Fl_Color labelcolor () const
  
  Gets the label color.

• void labelcolor (Fl_Color c)
  
  Sets the label color.

• Fl_Font labelfont () const
  
  Gets the font to use.

• void labelfont (Fl_Font f)
  
  Sets the font to use.

• Fl_Fontsize labelsize () const
Gets the font size in pixels.

• void *labelsize* (*Fl_Fontsize* pix)
  
  Sets the font size in pixels.

• *Fl_Labeltype* *labeltype* () const
  
  Gets the label type.

• void *labeltype* (*Fl_Labeltype* a)
  
  Sets the label type.

• void *measure_label* (int &ww, int &hh) const
  
  Sets width ww and height hh accordingly with the label size.

• bool *needs_keyboard* () const
  
  Returns whether this widget needs a keyboard.

• void *needs_keyboard* (bool needs)
  
  Sets whether this widget needs a keyboard.

• unsigned int *output* () const
  
  Returns if a widget is used for output only.

• *Fl_Group* ∗ *parent* () const
  
  Returns a pointer to the parent widget.

• void *parent* (*Fl_Group* ∗p)
  
  Internal use only - "for hacks only".

• void *position* (int X, int Y)
  
  Repositions the window or widget.

• void *redraw* ()
  
  Schedules the drawing of the widget.

• void *redraw_label* ()
  
  Schedules the drawing of the label.

• virtual void *resize* (int x, int y, int w, int h)
  
  Changes the size or position of the widget.

• *Fl_Color* *selection_color* () const
  
  Gets the selection color.

• void *selection_color* (*Fl_Color* a)
  
  Sets the selection color.

• void *set_active* ()
  
  Marks the widget as active without sending events or changing focus.

• void *set_changed* ()
  
  Marks the value of the widget as changed.

• void *set_output* ()
  
  Sets a widget to output only.

• void *set_visible* ()
  
  Makes the widget visible.

• void *set_visible_focus* ()
  
  Enables keyboard focus navigation with this widget.

• int *shortcut_label* () const
  
  Returns whether the widget's label uses '&' to indicate shortcuts.

• void *shortcut_label* (int value)
  
  Sets whether the widget's label uses '&' to indicate shortcuts.

• virtual void *show* ()
  
  Makes a widget visible.

• void *size* (int W, int H)
  
  Changes the size of the widget.

• int *take_focus* ()
  
  Gives the widget the keyboard focus.
• unsigned int takesevents () const

  Returns if the widget is able to take events.

• int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.

• const char * tooltip () const

  Gets the current tooltip text.

• void tooltip (const char *text)

  Sets the current tooltip text.

• Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.

• Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const

  Gets the widget type.

• void type (uchar t)

  Sets the widget type.

• int use_accents_menu ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void * user_data () const

  Gets the user data for this widget.

• void user_data (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.

• void user_data (void *v)

  Sets the user data for this widget.

• unsigned int visible () const

  Returns whether a widget is visible.

• unsigned int visible_focus () const

  Checks whether this widget has a visible focus.

• void visible_focus (int v)

  Modifies keyboard focus navigation.

• int visible_r () const

  Returns whether a widget and all its parents are visible.

• int w () const

  Gets the widget width.

• Fl_When when () const

  Returns the conditions under which the callback is called.

• void when (uchar i)

  Sets the flags used to decide when a callback is called.

• Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const

  Gets the widget position in its window.

• int y () const

  Gets the widget position in its window.

• virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

• void draw () FL_OVERRIDE

  Draws the widget.
Protected Member Functions inherited from Fl_Widget

- **void clear_flag** (unsigned int c)
  
  Clears a flag in the flags mask.

- **void draw_backdrop** () const
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- **void draw_box** () const
  
  Draws the widget box according its box style.

- **void draw_box** (Fl_Boxtype t, Fl_Color c) const
  
  Draws a box of type t, of color c at the widget's position and size.

- **void draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type t, of color c at the position X,Y and size W,H.

- **void draw_focus** () const
  
  Draws a focus rectangle around the widget.

- **void draw_focus** (Fl_Boxtype t, int X, int Y, int W, int H) const
  
  Draws a focus rectangle around the widget.

- **void draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.

- **void draw_label** () const
  
  Draws the widget's label at the defined label position.

- **void draw_label** (int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.

- **Fl_Widget** (int x, int y, int w, int h, const char *label=0L)
  
  Creates a widget at the given position and size.

- **unsigned int flags** () const
  
  Gets the widget flags mask.

- **void h** (int v)
  
  Internal use only.

- **void set_flag** (unsigned int c)
  
  Sets a flag in the flags mask.

- **void w** (int v)
  
  Internal use only.

- **void x** (int v)
  
  Internal use only.

- **void y** (int v)
  
  Internal use only.

**Additional Inherited Members**

**Static Public Member Functions inherited from Fl_Widget**

- **static void default_callback** (Fl_Widget *widget, void *data)
  
  The default callback for all widgets that don’t set a callback.

- **static unsigned int labelShortcut** (const char *t)
  
  Returns the Unicode value of the ‘&x’ shortcut in a given text.

- **static int testShortcut** (const char *, const bool require_alt=false)
  
  Returns true if the given text t contains the entered ‘&x’ shortcut.
Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6,Changed = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTOP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCEPTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

12.45.1 Member Function Documentation

12.45.1.1 draw()

void Fl_FormsText::draw ( ) [protected], [virtual]
Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as
soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded
scrollbar, you can do it (because draw() is virtual) like this:
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()

Implements Fl_Widget.
The documentation for this class was generated from the following file:

- forms.H

12.46 Fl_Free Class Reference

Emulation of the Forms "free" widget.
#include <Fl_Free.H>
Inheritance diagram for Fl_Free:

```
Fl_Widget
  ↓
Fl_Free
```

Public Member Functions

- **Fl_Free (uchar t, int X, int Y, int W, int H, const char *L, FL_HANDLEPTR hdl)**
  Create a new Fl_Free widget with type, position, size, label and handler.
- **int handle (int e) FL_OVERRIDE**
  Handles the specified event.
- **~Fl_Free ()**
  The destructor will call the handle function with the event FL_FREE_MEM.
Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  Activates the widget.
• unsigned int active () const
  Returns whether the widget is active.
• int active_r () const
  Returns whether the widget and all of its parents are active.
• Fl_Align align () const
  Gets the label alignment.
• void align (Fl_Align alignment)
  Sets the label alignment.
• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
• virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const ∗ as_gl_window () const
• virtual Fl_Group ∗ as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• virtual Fl_Group const ∗ as_group () const
• virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const ∗ as_window () const
• void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• Fl_Boxtype box () const
  Gets the box type of the widget.
• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
• Fl_Callback ∗ p callback () const
  Gets the current callback function for the widget.
• void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget ∗ w) const
  Checks if w is a child of this widget.
• void copy_label (const char ∗ new_label)
  Sets the current label.
• void copy_tooltip (const char ∗ text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image ∗ deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image ∗ deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image ∗ &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void *do_callback* (Fl_Widget *widget, long arg, Fl_Callback_Reas on reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void *do_callback* (Fl_Widget *widget, void *arg=0, Fl_Callback_Reas on reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void *draw_label* (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int *h* () const
  Gets the widget height.
• virtual void *hide* ()
  Makes a widget invisible.
• Fl_Image *image* ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image *image* () const
  Gets the image that is used as part of the widget label when in the active state.
• void *image* (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void *image* (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int *image_bound* () const
  Returns whether the image is managed by the widget.
• int *inside* (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int *is_label_copied* () const
  Returns whether the current label was assigned with copy_label().
• const char *label* () const
  Gets the current label text.
• void *label* (const char *text)
  Sets the current label pointer.
• void *label* (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color *labelcolor* () const
  Gets the label color.
• void *labelcolor* (Fl_Color c)
  Sets the label color.
• Fl_Font *labelfont* () const
  Gets the font to use.
• void *labelfont* (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize *labelsize* () const
  Gets the font size in pixels.
• void *labelsize* (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype *labeltype* () const
  Gets the label type.
• void *labeltype* (Fl_Labeltype a)
  Sets the label type.
• void *measure_label* (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool *needs_keyboard* () const
  Returns whether this widget needs a keyboard.
• void *needs_keyboard* (bool needs)
Sets whether this widget needs a keyboard.

- **unsigned int output()** const
  Returns if a widget is used for output only.

- **Fl_Group * parent()** const
  Returns a pointer to the parent widget.

- **void parent(Fl_Group *p)**
  Internal use only - "for hacks only".

- **void position(int X, int Y)**
  Repositions the window or widget.

- **void redraw()**
  Schedules the drawing of the widget.

- **void redraw_label()**
  Schedules the drawing of the label.

- **virtual void resize(int x, int y, int w, int h)**
  Changes the size or position of the widget.

- **Fl_Color selection_color()** const
  Gets the selection color.

- **void selection_color(Fl_Color a)**
  Sets the selection color.

- **void set_active()**
  Marks the widget as active without sending events or changing focus.

- **void set_changed()**
  Marks the value of the widget as changed.

- **void set_output()**
  Sets a widget to output only.

- **void set_visible()**
  Makes the widget visible.

- **void set_visible_focus()**
  Enables keyboard focus navigation with this widget.

- **int shortcut_label()** const
  Returns whether the widget's label uses '&' to indicate shortcuts.

- **void shortcut_label(int value)**
  Sets whether the widget's label uses '&' to indicate shortcuts.

- **virtual void show()**
  Makes a widget visible.

- **void size(int W, int H)**
  Changes the size of the widget.

- **int take_focus()**
  Gives the widget the keyboard focus.

- **unsigned int takeevents()** const
  Returns if the widget is able to take events.

- **int test_shortcut()**
  Returns true if the widget's label contains the entered '&x' shortcut.

- **const char * tooltip()** const
  Gets the current tooltip text.

- **void tooltip(const char *text)**
  Sets the current tooltip text.

- **Fl_Window * top_window()** const
  Returns a pointer to the top-level window for the widget.

- **Fl_Window * top_window_offset(int &xoff, int &yoff)** const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.

Protected Member Functions

• void draw () FL OVERRIDE
  Draws the widget.

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const
  Draws the widget box according its box style.
• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void draw_label () const
  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗ label=0L)
  Creates a widget at the given position and size.
- unsigned int flags () const
  Gets the widget flags mask.
- void h (int v)
  Internal use only.
- void set_flag (unsigned int c)
  Sets a flag in the flags mask.
- void w (int v)
  Internal use only.
- void x (int v)
  Internal use only.
- void y (int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗ widget, void ∗ data)
  Returns the default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char ∗ t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char ∗ t, const bool require_alt=false)
  Returns true if the given text t contains the entered 'x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22, AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
  flags possible values enumeration.
12.46.1 Detailed Description

Emulation of the Forms “free” widget. This emulation allows the free demo to run, and appears to be useful for porting programs written in Forms which use the free widget or make subclasses of the Forms widgets.

There are five types of free, which determine when the handle function is called:

- FL_NORMAL_FREE normal event handling.
- FL_SLEEPING_FREE deactivates event handling (widget is inactive).
- FL_INPUT_FREE accepts FL_FOCUS events.
- FL_CONTINUOUS_FREE sets a timeout callback 100 times a second and provides an FL_STEP event. This has obvious detrimental effects on machine performance.
- FL_ALL_FREE same as FL_INPUT_FREE and FL_CONTINUOUS_FREE.

12.46.2 Constructor & Destructor Documentation

12.46.2.1 Fl_Free()

Fl_Free::Fl_Free (uchar t, int X, int Y, int W, int H, const char * L, FL_HANDLEPTR hdl)

Create a new Fl_Free widget with type, position, size, label and handler.

Parameters

<table>
<thead>
<tr>
<th>in t</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>in X,Y,W,H</td>
<td>position and size</td>
</tr>
<tr>
<td>in L</td>
<td>widget label</td>
</tr>
<tr>
<td>in hdl</td>
<td>handler function</td>
</tr>
</tbody>
</table>

The constructor takes both the type and the handle function. The handle function should be declared as follows:

```c
int handle_function(Fl_Widget *w, int event, float event_x, float event_y, char key)
```

This function is called from the handle() method in response to most events, and is called by the draw() method.

The event argument contains the event type:

```c
#old event names for compatibility:
#define FL_MOUSE FL_DRAG
#define FL_DRAW 0
#define FL_STEP 9
#define FL_FREEMEM 12
#define FL_FREEZE FL_UNMAP
#define FL_THAW FL_MAP
```

12.46.3 Member Function Documentation

12.46.3.1 draw()

```c
void Fl_Free::draw () [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```c
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

### 12.46.3.2 handle()

```c
int Fl_Free::handle (int event) [virtual]
```

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class. In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

**Parameters**

- **in event** the kind of event received

**Return values**

- 0 if the event was not used or understood
- 1 if the event was used and can be deleted

**See also**

- Fl_Event

Reimplemented from Fl_Widget.

The documentation for this class was generated from the following files:

- Fl_Free.H
- forms_free.cxx

### 12.47 Fl_GIF_Image Class Reference

The Fl_GIF_Image class supports loading, caching, and drawing of Compuserve GIF\textsuperscript{SM} images.

```c
#include <Fl_GIF_Image.H>
```

Inheritance diagram for Fl_GIF_Image:

```
Fl_Image
   ↓
Fl_Pixmap
   ↓
Fl_GIF_Image
   ↓
Fl_Anim_GIF_Image
```

Generated by Doxygen
Classes

- struct GIF_FRAME

Public Member Functions

- **FL_GIF_Image (const char *filename)**
  
  This constructor loads a GIF image from the given file.

- **FL_GIF_Image (const char *imagename, const unsigned char *data)**
  
  This constructor loads a GIF image from memory (deprecated).

- **FL_GIF_Image (const char *imagename, const unsigned char *data, const size_t length)**
  
  This constructor loads a GIF image from memory.

Public Member Functions inherited from Fl_Pixmap

- int cache_h ()
- int cache_w ()
- void color_average (Fl_Color c, float i) FL_OVERRIDE
  
  The color_average() method averages the colors in the image with the provided FLTK color value.

- Fl_Image * copy () const

- Fl_Image * copy (int W, int H) const FL_OVERRIDE
  
  Creates a resized copy of the image.

- void desaturate () FL_OVERRIDE
  
  The desaturate() method converts an image to grayscale.

- void draw (int X, int Y)

- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE
  
  Draws the image to the current drawing surface with a bounding box.

- Fl_Pixmap (char *const *D)
  
  The constructors create a new pixmap from the specified XPM data.

- Fl_Pixmap (const char *const *D)
  
  The constructors create a new pixmap from the specified XPM data.

- Fl_Pixmap (const uchar *const *D)
  
  The constructors create a new pixmap from the specified XPM data.

- Fl_Pixmap (uchar *const *D)
  
  The constructors create a new pixmap from the specified XPM data.

- void label (Fl_Menu_Item *m) FL_OVERRIDE
  
  This method is an obsolete way to set the image attribute of a menu item.

- void label (Fl_Widget *w) FL_OVERRIDE
  
  This method is an obsolete way to set the image attribute of a widget or menu item.

- void uncache () FL_OVERRIDE
  
  If the image has been cached for display, delete the cache data.

- virtual ~Fl_Pixmap ()
  
  The destructor frees all memory and server resources that are used by the pixmap.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image * as_shared_image ()
  
  Returns whether an image is an Fl_Shared_Image or not.

- Fl_Image * copy () const
  
  Creates a copy of the image in the same size.

- int count () const
  
  Returns the number of data values associated with the image.

- int d () const
Returns the image depth.

- const char *const * data () const
  
  Returns a pointer to the current image data array.

- int data_h () const
  
  Returns the height of the image data.

- int data_w () const
  
  Returns the width of the image data.

- void draw (int X, int Y)
  
  Draws the image to the current drawing surface.

- int fail () const
  
  Returns a value that is not 0 if there is currently no image available.

- Fl_Image (int W, int H, int D)
  
  The constructor creates an empty image with the specified width, height, and depth.

- int h () const
  
  Returns the current image drawing height in FLTK units.

- void inactive ()
  
  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

- int ld () const
  
  Returns the current line data size in bytes.

- virtual void release ()
  
  Releases an Fl_Image - the same as 'delete this'.

- virtual void scale (int width, int height, int proportional=1, int can_expand=0)
  
  Sets the drawing size of the image.

- int w () const
  
  Returns the current image drawing width in FLTK units.

- virtual ~Fl_Image ()
  
  The destructor is a virtual method that frees all memory used by the image.

Static Public Member Functions

- static bool is_animated (const char *name_)

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype define_FL_IMAGE_LABEL ()

- static Fl_RGB_Scaling RGB_scaling ()
  
  Returns the currently used RGB image scaling method.

- static void RGB_scaling (Fl_RGB_Scaling)
  
  Sets the RGB image scaling method used for copy(int, int).

- static Fl_RGB_Scaling scaling_algorithm ()
  
  Gets what algorithm is used when resizing a source image to draw it.

- static void scaling_algorithm (Fl_RGB_Scaling algorithm)
  
  Sets what algorithm is used when resizing a source image to draw it.

Static Public Attributes

- static bool animate = false
  
  Sets how the shared image core routine should treat animated GIF files.

Generated by Doxygen
Static Public Attributes inherited from Fl_Image

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_MEMORY_ACCESS = -4
- static const int ERR_NO_IMAGE = -1
- static bool register_images_done = false

Protected Member Functions

- Fl_GIF_Image ()
  The default constructor creates an empty GIF image.
- Fl_GIF_Image (const char ∗filename, bool anim)
- Fl_GIF_Image (const char ∗imagename, const unsigned char ∗data, const size_t length, bool anim)
- void load (const char ∗filename, bool anim)
- void load (const char ∗imagename, const unsigned char ∗data, const size_t length, bool anim)
- void load_gif_ (class Fl_Image_Reader &rdr, bool anim=false)
- virtual void on_extension_data (GIF_FRAME &)
- virtual void on_frame_data (GIF_FRAME &)

Protected Member Functions inherited from Fl_Pixmap

- void measure ()

Protected Member Functions inherited from Fl_Image

- void d (int D)
  Sets the current image depth.
- void data (const char ∗const ∗p, int c)
  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)
  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)
  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)
  Sets the height of the image data.
- void ld (int LD)
  Sets the current line data size in bytes.
- void w (int W)
  Sets the width of the image data.

Additional Inherited Members

Public Attributes inherited from Fl_Pixmap

- int alloc_data

Static Protected Member Functions inherited from Fl_Image

- static void labeltype (const Fl_Label ∗lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void measure (const Fl_Label ∗lo, int &lw, int &lh)
12.47.1 Detailed Description

The Fl_GIF_Image class supports loading, caching, and drawing of Compuserve GIF images. The class loads the first image and supports transparency.

12.47.2 Constructor & Destructor Documentation

12.47.2.1 Fl_GIF_Image() [1/3]

Fl_GIF_Image::Fl_GIF_Image (const char * filename)

This constructor loads a GIF image from the given file. If a GIF image is animated, Fl_GIF_Image will only read and display the first frame of the animation. The destructor frees all memory and server resources that are used by the image. Use Fl_Image::fail() to check if Fl_GIF_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the GIF format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

| in | filename | a full path and name pointing to a GIF image file. |

See also

Fl_GIF_Image::Fl_GIF_Image(const char * imagename, const unsigned char * data, const long length)

12.47.2.2 Fl_GIF_Image() [2/3]

Fl_GIF_Image::Fl_GIF_Image (const char * imagename, const unsigned char * data)

This constructor loads a GIF image from memory (deprecated).

Deprecated Please use Fl_GIF_Image(const char * imagename, const unsigned char * data, const size_t length) instead.

Note

Buffer overruns will not be checked.

This constructor should not be used because the caller can't supply the memory size and the image reader can't check for "end of memory" errors.

Note

A new constructor with parameter length is available since FLTK 1.4.0.

Parameters

| in | imagename | A name given to this image or NULL |
| in | data | Pointer to the start of the GIF image in memory. |

See also

Fl_GIF_Image(const char * filename)

Fl_GIF_Image(const char * imagename, const unsigned char * data, const size_t length)
12.47.2.3 Fl_GIF_Image() [3/3]

Fl_GIF_Image::Fl_GIF_Image (const char * imagename, const unsigned char * data, const size_t length)

This constructor loads a GIF image from memory. Construct an image from a block of memory inside the application. Fluid offers "binary data" chunks as a great way to add image data into the C++ source code. imagename can be NULL. If a name is given, the image is added to the list of shared images and will be available by that name.

If a GIF image is animated, Fl_GIF_Image will only read and display the first frame of the animation. The destructor frees all memory and server resources that are used by the image. The third parameter length is used to test for buffer overruns, i.e. truncated images. Use Fl_Image::fail() to check if Fl_GIF_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the GIF format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imagename</td>
<td>A name given to this image or NULL</td>
</tr>
<tr>
<td>data</td>
<td>Pointer to the start of the GIF image in memory.</td>
</tr>
<tr>
<td>length</td>
<td>Length of the GIF image in memory.</td>
</tr>
</tbody>
</table>

See also

    Fl_GIF_Image::Fl_GIF_Image(const char *filename)
    Fl_Shared_Image

Since

    1.4.0

12.47.3 Member Data Documentation

12.47.3.1 animate

bool Fl_GIF_Image::animate = false [static]

Sets how the shared image core routine should treat animated GIF files. The default is to treat them as ordinary GIF's e.g. it creates a Fl_GIF_Image object. If this variable is set, then an animated GIF object Fl_Anim_GIF_Image is created.

The documentation for this class was generated from the following files:

- Fl_GIF_Image.H
- Fl_Anim_GIF_Image.cxx
- Fl_GIF_Image.cxx

12.48 Fl_GI_Choice Class Reference

Public Member Functions

- Fl_GI_Choice (int m, const int *alistp, Fl_GI_Choice *n)

Friends

- class Fl_GI_Window_Driver

The documentation for this class was generated from the following file:

- Fl_GI_Choice.H
12.49  Fl_Gl_Window Class Reference

The Fl_Gl_Window widget sets things up so OpenGL works.

```cpp
#include <Fl_Gl_Window.H>
```

Inheritance diagram for Fl_Gl_Window:

```
Fl_Widget
    ↓
Fl_Group
    ↓
Fl_Window
    ↓
Fl_Gl_Window
    ↓
Fl_Glut_Window
```

Public Member Functions

- `Fl_Gl_Window const * as_gl_window () const FL_OVERRIDE`
  - Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- `int can_do ()`
  - Returns non-zero if the hardware supports the current OpenGL mode.
- `int can_do_overlay ()`
  - Returns true if the hardware overlay is possible.
- `GLContext context () const`
  - Returns a pointer to the window's OpenGL rendering context.
- `void context (GLContext, int destroy_flag=0)`
  - Sets a pointer to the GLContext that this window is using.
- `char context_valid () const`
  - Will only be set if the OpenGL context is created or recreated.
- `void context_valid (char v)`
  - See char Fl_Gl_Window::context_valid() const.
- `Fl_Gl_Window (int W, int H, const char *l=0)`
  - Creates a new Fl_Gl_Window widget using the given size, and label string.
- `Fl_Gl_Window (int X, int Y , int W, int H, const char *l=0)`
  - Creates a new Fl_Gl_Window widget using the given position, size, and label string.
- `void flush () FL_OVERRIDE`
  - Forces the window to be drawn, this window is also made current and calls draw().
- `int handle (int) FL_OVERRIDE`
  - Handle some FLTK events as needed.
- `void hide () FL_OVERRIDE`
  - Hides the window and destroys the OpenGL context.
- `void hide_overlay ()`
  - Hides the window if it is not this window, does nothing in Windows.
- `void invalidate ()`
  - The invalidate() method turns off valid() and is equivalent to calling value(0).
- `void make_current ()`
  - The make_current() method selects the OpenGL context for the widget.
- `void make_overlay_current ()`
Selects the OpenGL context for the widget’s overlay.

- **Fl_Mode mode () const**
  Returns the current OpenGL capabilities of the window.

- **int mode (const int *a)**
  Set the OpenGL capabilities of the window using platform-specific data.

- **int mode (int a)**
  Set or change the OpenGL capabilities of the window.

- **void ortho ()**
  Sets the projection so 0,0 is in the lower left of the window and each pixel is 1 unit wide/tall.

- **int pixel_h ()**
  Gives the window height in OpenGL pixels.

- **int pixel_w ()**
  Gives the window width in OpenGL pixels.

- **float pixels_per_unit ()**
  The number of pixels per FLTK unit of length for the window.

- **void redraw_overlay ()**
  Causes draw_overlay() to be called at a later time.

- **void resize (int, int, int, int) FL_OVERRIDE**
  Changes the size or position of the widget.

- **void show () FL_OVERRIDE**
  Makes a widget visible.

- **void show (int a, char **b)**
  Same as Fl_Window::show(int a, char **b)

- **void swapBuffers ()**
  The swapBuffers() method swaps the back and front buffers.

- **int swap_interval () const**
  Gets the rate at which the GL windows swaps buffers.

- **void swap_interval (int)**
  Sets the rate at which the GL windows swaps buffers.

- **char valid () const**
  Is turned off when FLTK creates a new context for this window or when the window resizes, and is turned on after draw() is called.

- **void valid (char v)**
  See char Fl_Gl_Window::valid() const.

- **Fl_Gl_Window ()**
  The destructor removes the widget and destroys the OpenGL context associated with it.

## Public Member Functions inherited from Fl_Window

- **virtual class Fl_Double_Window * as_double_window ()**
  Return non-null if this is an Fl_Double_Window object.

- **virtual class Fl_Overlay_Window * as_overlay_window ()**
  Return non-null if this is an Fl_Overlay_Window object.

- **Fl_Window const * as_window () const FL_OVERRIDE**
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- **unsigned int border () const**
  Returns whether the window possesses a border.

- **void border (int b)**
  Sets whether or not the window manager border is around the window.

- **void clear_border ()**
Fast inline function to turn the window manager border off.

- void **clear_modal_states** ()
  
  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.

- void **copy_label** (const char ∗a)
  
  Sets the window title bar label to a copy of a character string.

- void **cursor** (const Fl_RGB_Image ∗, int, int)
  
  Changes the cursor for this window using the provided image as cursor's shape.

- void **cursor** (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  
  For back compatibility only.

- void **cursor** (Fl_Cursor)
  
  Changes the cursor for this window.

- int **decorated_h** () const
  
  Returns the window height including any window title bar and any frame added by the window manager.

- int **decorated_w** () const
  
  Returns the window width including any frame added by the window manager.

- void **default_cursor** (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  
  For back compatibility only.

- void **default_cursor** (Fl_Cursor)
  
  Sets the default window cursor.

- void **draw_backdrop** ()
  
  Draw the background image if one is set and is aligned inside.

- Fl_Window (int w, int h, const char ∗title=0)
  
  Creates a window from the given width \( w \), height \( h \), and \( title \).

- Fl_Window (int x, int y, int w, int h, const char ∗title=0)
  
  Creates a window from the given position \( (x, y) \), size \( (w, h) \) and \( title \).

- void **free_position** ()
  
  Undoes the effect of a previous \( \text{resize()} \) or \( \text{show()} \) so that the next time \( \text{show()} \) is called the window manager is free to position the window.

- void **fullscreen** ()
  
  Makes the window completely fill one or more screens, without any window manager border visible.

- unsigned int **fullscreen_active** () const
  
  Returns non zero if \( \text{FULLSCREEN} \) flag is set, 0 otherwise.

- void **fullscreen_off** ()
  
  Turns off any side effects of \( \text{fullscreen()} \)

- void **fullscreen_off** (int X, int Y, int W, int H)
  
  Turns off any side effects of \( \text{fullscreen()} \) and does \( \text{resize(x,y,w,h)} \).

- void **fullscreen_screens** (int top, int bottom, int left, int right)
  
  Sets which screens should be used when this window is in fullscreen mode.

- int **handle** (int) FL_OVERRIDE
  
  Handles the specified event.

- void **hide** () FL_OVERRIDE
  
  Removes the window from the screen.

- void **hotspot** (const Fl_Widget &p, int offscreen=0)
  
  See void \( \text{Fl_Window::hotspot(int x, int y, int offscreen = 0)} \)

- void **hotspot** (const Fl_Widget ∗, int offscreen=0)
  
  See void \( \text{Fl_Window::hotspot(int x, int y, int offscreen = 0)} \)

- void **hotspot** (int x, int y, int offscreen=0)
  
  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.

- const void ∗**icon** () const
  
  Gets the current icon window target dependent data.
• void **icon** (const **Fl_RGB_Image** *)
  Sets or resets a single window icon.
• void **icon** (const void *)
  Platform-specific method to set the window icon usable on Windows and X11 only.
• void **iconize** ()
  Iconifies the window.
• const char * **iconlabel** () const
  See void **Fl_Window::iconlabel**(const char*)
• void **iconlabel** (const char *)
  Sets the icon label.
• void **icons** (const **Fl_RGB_Image** *[], int)
  Sets the window icons.
• void **icons** (**HICON** big_icon, **HICON** small_icon)
  Sets the window icons using HICON handles (Windows platform only).
• const char * **label** () const
  See void **Fl_Window::label**(const char*)
• void **label** (const char *)
  Sets the window title bar label.
• void **label** (const char *label, const char *iconlabel)
  Sets the icon label.
• void **make_current** ()
  Sets things up so that the drawing functions in <**FL/fl_draw.H**> will go into this window.
• void **maximize** ()
  Maximizes a top-level window to its current screen.
• unsigned int **maximize_active** () const
  Returns whether the window is currently maximized.
• unsigned int **menu_window** () const
  Returns true if this window is a menu window.
• unsigned int **modal** () const
  Returns true if this window is modal.
• unsigned int **non_modal** () const
  Returns true if this window is modal or non-modal.
• **fluintptr_t** **os_id** ()
  Returns a platform-specific identification of a shown window, or 0 if not shown.
• unsigned int **override** () const
  Returns non zero if OVERRIDE flag is set, 0 otherwise.
• void **resize** (int X, int Y, int W, int H) **FL_OVERRIDE**
  Changes the size and position of the window.
• int **screen_num** ()
  The number of the screen containing the mapped window.
• void **screen_num** (int screen_num)
  Set the number of the screen where to map the window.
• void **set_menu_window** ()
  Marks the window as a menu window.
• void **set_modal** ()
  A "modal" window, when **shown()**, will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).
• void **set_non_modal** ()
A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.

- **void set_override ()**
  Activates the flags NOBORDER|OVERRIDE.
- **void set_tooltip_window ()**
  Marks the window as a tooltip window.
- **const Fl_Image ∗ shape ()**
  Returns the image controlling the window shape or NULL.
- **void shape (const Fl_Image &b)**
  Set the window’s shape with an Fl_Image.
- **void shape (const Fl_Image ∗img)**
  Assigns a non-rectangular shape to the window.
- **void show () FL_OVERRIDE**
  Puts the window on the screen.
- **void show (int argc, char ∗∗argv)**
  Puts the window on the screen with show() and parses command-line arguments.
- **int shown ()**
  Returns non-zero if show() has been called (but not hide() ).
- **void size_range (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)**
  Sets the allowable range the user can resize this window to.
- **unsigned int tooltip_window () const**
  Returns true if this window is a tooltip window.
- **void un_maximize ()**
  Returns a previously maximized top-level window to its previous size.
- **void wait_for_expose ()**
  Waits for the window to be displayed after calling show().
- **int x_root () const**
  Gets the x position of the window on the screen.
- **const char ∗ xclass () const**
  Returns the xclass for this window, or a default.
- **void xclass (const char ∗c)**
  Sets the xclass for this window.
- **int y_root () const**
  Gets the y position of the window on the screen.
- **virtual ∼Fl_Window ()**
  The destructor also deletes all the children.

**Public Member Functions inherited from Fl_Group**

- **Fl_Widget ∗& _ddfdesign_kludge ()**
  This is for forms compatibility only.
- **void add (Fl_Widget &)**
  The widget is removed from its current group (if any) and then added to the end of this group.
- **void add (Fl_Widget ∗o)**
  See void Fl_Group::add(Fl_Widget &w)
- **void add_resizable (Fl_Widget &o)**
  Adds a widget to the group and makes it the resizable widget.
- **Fl_Widget ∗const ∗ array () const**
  Returns a pointer to the array of children.
- **Fl_Group const ∗ as_group () const FL_OVERRIDE**
- **Fl_Group ∗ as_group () FL_OVERRIDE**
Returns an Fl_Group pointer if this widget is an Fl_Group.

- **void begin ()**
  Sets the current group so you can build the widget tree by just constructing the widgets.

- **Fl_Widget ∗ child (int n) const**
  Returns arra[y] [n].

- **int children () const**
  Returns how many child widgets the group has.

- **void clear ()**
  Deletes all child widgets from memory recursively.

- **unsigned int clip_children ()**
  Returns the current clipping mode.

- **void clip_children (int c)**
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- **virtual int delete_child (int n)**
  Removes the widget at index from the group and deletes it.

- **void end ()**
  Exactly the same as current(this->parent()).

- **int find (const Fl_Widget &o) const**
  See int Fl_Group::find(const Fl_Widget ∗w) const.

- **int find (const Fl_Widget ∗) const**
  Searches the child array for the widget and returns the index.

- **Fl_Group (int, int, int, int, const char ∗=0)**
  Creates a new Fl_Group widget using the given position, size, and label string.

- **void focus (Fl_Widget ∗W)**

- **void forms_end ()**
  This is for forms compatibility only.

- **void init_sizes ()**
  Resets the internal array of widget sizes and positions.

- **void insert (Fl_Widget &, int i)**
  The widget is removed from its current group (if any) and then inserted into this group.

- **void insert (Fl_Widget &, Fl_Widget ∗before)**
  This does insert(w, find(before)).

- **void remove (Fl_Widget ∗) &**
  Removes a widget from the group but does not delete it.

- **void remove (Fl_Widget ∗o)**
  Removes the widget o from the group.

- **void remove (int index)**
  Removes the widget at index from the group but does not delete it.

- **Fl_Widget ∗ resizable () const**
  Returns the group's resizable widget.

- **void resizable (Fl_Widget ∗o)**
  Sets the group's resizable widget.

- **void resizable (Fl_Widget ∗o)**
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- **virtual ∼Fl_Group ()**
  The destructor also deletes all the children.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- void bind_deimage (Fl_Image img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback cb, Fl_Callback_User_Data p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback cb, void p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
  Checks if the widget value changed since the last callback.
- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()
  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)
  Clears or sets the damage flags.
- void clear_output ()
Sets a widget to accept input.

- **void clear_visible ()**
  
  Hides the widget.

- **void clear_visible_focus ()**
  
  Disables keyboard focus navigation with this widget.

- **Fl_Color color () const**
  
  Gets the background color of the widget.

- **void color (Fl_Color bg)**
  
  Sets the background color of the widget.

- **void color (Fl_Color bg, Fl_Color sel)**
  
  Sets the background and selection color of the widget.

- **Fl_Color color2 () const**
  
  For back compatibility only.

- **void color2 (unsigned a)**
  
  For back compatibility only.

- **int contains (const Fl_Widget *w) const**
  
  Checks if w is a child of this widget.

- **void copy_label (const char *new_label)**
  
  Sets the current label.

- **void copy_tooltip (const char *text)**
  
  Sets the current tooltip text.

- **uchar damage () const**
  
  Returns non-zero if draw() needs to be called.

- **void damage (uchar c)**
  
  Sets the damage bits for the widget.

- **void damage (uchar c, int x, int y, int w, int h)**
  
  Sets the damage bits for an area inside the widget.

- **int damage_resize (int, int, int, int)**
  
  Internal use only.

- **void deactivate ()**
  
  Deactivates the widget.

- **Fl_Image * deimage ()**
  
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image * deimage () const**
  
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (Fl_Image &img)**
  
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (Fl_Image *img)**
  
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const**
  
  Returns whether the inactive image is managed by the widget.

- **void do_callback (Fl_CallbackReason reason=FL_REASON_UNKNOWN)**
  
  Calls the widget callback function with default arguments.

- **void do_callback (Fl_Widget +widget, long arg, Fl_CallbackReason reason=FL_REASON_UNKNOWN)**
  
  Calls the widget callback function with arbitrary arguments.

- **void do_callback (Fl_Widget +widget, void +arg=0, Fl_CallbackReason reason=FL_REASON_UNKNOWN)**
  
  Calls the widget callback function with arbitrary arguments.

- **void draw_label (int, int, int, Fl_Align) const**
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h () const**
  
  Gets the widget height.
• `Fl_Image * image ()`
  Gets the image that is used as part of the widget label when in the active state.

• `const Fl_Image * image () const`
  Gets the image that is used as part of the widget label when in the active state.

• `void image (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the active state.

• `void image (Fl_Image * img)`
  Sets the image to use as part of the widget label when in the active state.

• `int image_bound () const`
  Returns whether the image is managed by the widget.

• `int inside (const Fl_Widget * wgt) const`
  Checks if this widget is a child of `wgt`.

• `int is_label_copied () const`
  Returns whether the current label was assigned with `copy_label()`.

• `const char * label () const`
  Gets the current label text.

• `void label (const char * text)`
  Sets the current label pointer.

• `void label (Fl_Labeltype a, const char * b)`
  Shortcut to set the label text and type in one call.

• `Fl_Color labelcolor () const`
  Gets the label color.

• `void labelcolor (Fl_Color c)`
  Sets the label color.

• `Fl_Font labelfont () const`
  Gets the font to use.

• `void labelfont (Fl_Font f)`
  Sets the font to use.

• `Fl_Fontsize labelsize () const`
  Gets the font size in pixels.

• `void labelsize (Fl_Fontsize pix)`
  Sets the font size in pixels.

• `Fl_Labeltype labeltype () const`
  Gets the label type.

• `void labeltype (Fl_Labeltype a)`
  Sets the label type.

• `void measure_label (int &ww, int &hh) const`
  Sets width `ww` and height `hh` accordingly with the label size.

• `bool needs_keyboard () const`
  Returns whether this widget needs a keyboard.

• `void needs_keyboard (bool needs)`
  Sets whether this widget needs a keyboard.

• `unsigned int output () const`
  Returns if a widget is used for output only.

• `Fl_Group * parent () const`
  Returns a pointer to the parent widget.

• `void parent (Fl_Group * p)`
  Internal use only - “for hacks only”.

• `void position (int X, int Y)`
  Repositions the window or widget.

• `void redraw ()`
Schedules the drawing of the widget.

- **void **redraw_label ()
  Schedules the drawing of the label.

- **Fl_Color** selection_color () const
  Gets the selection color.

- **void** selection_color (Fl_Color a)
  Sets the selection color.

- **void** set_active ()
  Marks the widget as active without sending events or changing focus.

- **void** set_changed ()
  Marks the value of the widget as changed.

- **void** set_output ()
  Sets a widget to output only.

- **void** set_visible ()
  Makes the widget visible.

- **void** set_visible_focus ()
  Enables keyboard focus navigation with this widget.

- **int** shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.

- **void** shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

- **void** size (int W, int H)
  Changes the size of the widget.

- **int** take_focus ()
  Gives the widget the keyboard focus.

- **unsigned int** takesevents () const
  Returns if the widget is able to take events.

- **int** test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.

- **const char ∗** tooltip () const
  Gets the current tooltip text.

- **void** tooltip (const char ∗text)
  Sets the current tooltip text.

- **Fl_Window ∗** top_window () const
  Returns a pointer to the top-level window for the widget.

- **Fl_Window ∗** top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

- **uchar** type () const
  Gets the widget type.

- **void** type (uchar t)
  Sets the widget type.

- **int** use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- **void ∗** user_data () const
  Gets the user data for this widget.

- **void** user_data (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.

- **void** user_data (void ∗v)
  Sets the user data for this widget.

- **unsigned int** visible () const
  Returns whether a widget is visible.
- unsigned int **visible_focus** () const
  Checks whether this widget has a visible focus.
- void **visible_focus** (int v)
  Modifies keyboard focus navigation.
- int **visible_r** () const
  Returns whether a widget and all its parents are visible.
- int **w** () const
  Gets the widget width.
- **Fl_When** **when** () const
  Returns the conditions under which the callback is called.
- void **when** (uchar i)
  Sets the flags used to decide when a callback is called.
- **Fl_Window** *window** () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
- int **x** () const
  Gets the widget position in its window.
- int **y** () const
  Gets the widget position in its window.
- virtual ~**Fl_Widget** ()
  Destroys the widget.

**Static Public Member Functions**

- static int **can_do** (const int *m)
  Returns non-zero if the hardware supports the given OpenGL mode.
- static int **can_do** (int m)
  Returns non-zero if the hardware supports the given OpenGL mode.

**Static Public Member Functions inherited from Fl_Window**

- static **Fl_Window** *current** ()
  Returns the last window that was made current.
- static void **default_callback** (Fl_Window *, void *)
  Back compatibility: Sets the default callback v for win to call on close event.
- static void **default_icon** (const Fl_RGB_Image *v)
  Sets a single default window icon.
- static void **default_icons** (const Fl_RGB_Image *[ ]; int)
  Sets the default window icons.
- static void **default_icons** (HICON big_icon, HICON small_icon)
  Sets the default window icons (Windows platform only).
- static const char * **default_xclass** ()
  Returns the default xclass.
- static void **default_xclass** (const char *v)
  Sets the default window xclass.
- static bool **is_a_rescale** ()
  Returns true when a window is being rescaled.
- static char **show_next_window_iconic** ()
  Returns the static flag whether the next window should be opened iconified.
- static void **show_next_window_iconic** (char stat)
  Sets a static flag whether the next window should be opened iconified.
Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()
  Returns the currently active group.
- static void current (Fl_Group *g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don’t set a callback.
- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the ‘&x’ shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered ‘&x’ shortcut.

Protected Member Functions

- void draw () FL_OVERRIDE
  Draws the Fl_Gl_Window.
- void draw_begin ()
  Supports drawing to an Fl_Gl_Window with the FLTK 2D drawing API.
- void draw_end ()
  To be used as a match for a previous call to Fl_Gl_Window::draw_begin().

Protected Member Functions inherited from Fl_Window

- void default_size_range ()
  Protected method to calculate the default size range of a window.
- void draw () FL_OVERRIDE
  Draws the widget.
- int force_position () const
  Returns the internal state of the window’s FORCE_POSITION flag.
- void force_position (int force)
  Sets an internal flag that tells FLTK and the window manager to honor position requests.
- void free_icons ()
  Deletes all icons previously attached to the window.
- int is_resizable ()
  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.
- void draw_children ()
  Draws all children of the group.
- void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)
Allow derived groups to act when a widget is moved within the group.
• virtual void on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.
• int * sizes ()
  Returns the internal array of widget sizes and positions.
• void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const
  Draws the widget box according its box style.
• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
• void draw_focus () const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
• void draw_label () const
  Draws the widget's label at the defined label position.
• void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
• Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.
• unsigned int flags () const
  Gets the widget flags mask.
• void h (int v)
  Internal use only.
• void set_flag (unsigned int c)
  Sets a flag in the flags mask.
• void w (int v)
  Internal use only.
• void x (int v)
  Internal use only.
• void y (int v)
  Internal use only.

Friends

• class Fl_Gl_Window_Driver

Additional Inherited Members

Public Types inherited from Fl_Window

• typedef struct HICON__ * HICON
Protected Types inherited from Fl_Widget

```
enum {
    INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
    FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
    OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
    MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
    GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
    = 1<<19, NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
    AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.
```

Static Protected Attributes inherited from Fl_Window

```
• static Fl_Window * current_
  Stores the last window that was made current.
```

12.49.1 Detailed Description

The Fl_Gl_Window widget sets things up so OpenGL works. It also keeps an OpenGL "context" for that window, so that changes to the lighting and projection may be reused between redraws. Fl_Gl_Window also flushes the OpenGL streams and swaps buffers after draw() returns. OpenGL hardware typically provides some overlay bit planes, which are very useful for drawing UI controls atop your 3D graphics. If the overlay hardware is not provided, FLTK tries to simulate the overlay. This works pretty well if your graphics are double buffered, but not very well for single-buffered. Please note that the FLTK drawing and clipping functions will not work inside an Fl_Gl_Window. All drawing should be done using OpenGL calls exclusively.

See also

OpenGL and support of HighDPI displays

Note

FLTK 1.4 introduces a driver system for graphic calls. It is now possible to add a selection of widgets to an OpenGL window. The widgets will draw on top of any OpenGL rendering. The number of supported widgets will increase as the driver development improves. Program test/cube.cxx illustrates how to do that. FLTK expects that when an Fl_Gl_Window is a child of a parent Fl_Window, the child window lies entirely inside its parent window. If that's not the case, what happens to the part of the GL subwindow which leaks outside its parent is undefined and susceptible to be platform-specific.

12.49.2 Constructor & Destructor Documentation

12.49.2.1 Fl_Gl_Window() [1/2]

```
Fl_Gl_Window::Fl_Gl_Window ( int W, int H, const char * l = 0 ) [inline]
```

Creates a new Fl_Gl_Window widget using the given size, and label string. The default boxtype is FL_NO_BOX. The default mode is FL_RGB|FL_DOUBLE|FL_DEPTH.
12.49.2.2 \texttt{Fl\_Gl\_Window()} [2/2]

\begin{verbatim}
Fl_Gl_Window::Fl_Gl_Window (  
    int \texttt{X},  
    int \texttt{Y},  
    int \texttt{W},  
    int \texttt{H},  
    const char * \texttt{l} = 0 ) \{ \textrm{inline} \}
\end{verbatim}

Creates a new \texttt{Fl\_Gl\_Window} widget using the given position, size, and label string.
The default boxtype is \texttt{FL\_NO\_BOX}. The default mode is \texttt{FL\_RGB|FL\_DOUBLE|FL\_DEPTH}.

12.49.3 Member Function Documentation

12.49.3.1 \texttt{as\_gl\_window()} [1/2]

\begin{verbatim}
Fl_Gl_Window const * Fl_Gl_Window::as_gl_window ( ) const \{ \textrm{inline}, \textrm{virtual} \}
\end{verbatim}
Reimplemented from \texttt{Fl\_Widget}.

12.49.3.2 \texttt{as\_gl\_window()} [2/2]

\begin{verbatim}
Fl_Gl_Window * Fl_Gl_Window::as_gl_window ( ) \{ \textrm{inline}, \textrm{virtual} \}
\end{verbatim}
Returns an \texttt{Fl\_Gl\_Window} pointer if this widget is an \texttt{Fl\_Gl\_Window}.
Use this method if you have a widget (pointer) and need to know whether this widget is derived from \texttt{Fl\_Gl\_Window}.
If it returns non-NULL, then the widget in question is derived from \texttt{Fl\_Gl\_Window}.

Return values

\begin{verbatim}
\texttt{NULL} if this widget is not derived from \texttt{Fl\_Gl\_Window}.
\end{verbatim}

Note

This method is provided to avoid \texttt{dynamic\_cast}.

See also

\texttt{Fl\_Widget::as\_group()}, \texttt{Fl\_Widget::as\_window()}

Reimplemented from \texttt{Fl\_Widget}.

12.49.3.3 \texttt{can\_do()}

\begin{verbatim}
static int Fl_Gl_Window::can_do (  
    const int * \texttt{m} ) \{ \textrm{inline}, \textrm{static} \}
\end{verbatim}
Returns non-zero if the hardware supports the given OpenGL mode.

See also

\texttt{Fl\_Gl\_Window::mode(const int \texttt{m})}

12.49.3.4 \texttt{can\_do\_overlay()}

\begin{verbatim}
int Fl_Gl_Window::can_do_overlay ( )
\end{verbatim}
Returns true if the hardware overlay is possible.
If this is false, FLTK will try to simulate the overlay, with significant loss of update speed. Calling this will cause FLTK to open the display.

12.49.3.5 \texttt{context()} [1/2]

\begin{verbatim}
GLContext Fl_Gl_Window::context ( ) const \{ \textrm{inline} \}
\end{verbatim}
Returns a pointer to the window's OpenGL rendering context.
12.49.3.6 context() [2/2]

```cpp
void Fl_Gl_Window::context (GLContext v, int destroy_flag = 0)
```

Sets a pointer to the GLContext that this window is using. This is a system-dependent structure, but it is portable to copy the context from one window to another. You can also set it to NULL, which will force FLTK to recreate the context the next time `make_current()` is called, this is useful for getting around bugs in OpenGL implementations.

If `destroy_flag` is true the context will be destroyed by fltk when the window is destroyed, or when the mode() is changed, or the next time context(x) is called.

12.49.3.7 context_valid()

```cpp
char Fl_Gl_Window::context_valid () const [inline]
```

Will only be set if the OpenGL context is created or recreated.

It differs from `Fl_Gl_Window::valid()` which is also set whenever the context changes size.

12.49.3.8 draw()

```cpp
void Fl_Gl_Window::draw () [protected], [virtual]
```

Draws the `Fl_Gl_Window`.

You **must** subclass `Fl_Gl_Window` and provide an implementation for `draw()`. You may also provide an implementation of `draw_overlay()` if you want to draw into the overlay planes. You can avoid reinitializing the viewport and lights and other things by checking `valid()` at the start of `draw()` and only doing the initialization if it is false.

The `draw()` method can only use OpenGL calls. Do not attempt to call X, any of the functions in `<FL/fl_draw.H>`, or glX directly. Do not call `gl_start()` or `gl_finish()`.

If double-buffering is enabled in the window, the back and front buffers are swapped after this function is completed.

The following pseudo-code shows how to use "if (valid())" to initialize the viewport:

```cpp
void mywindow::draw() {
  if (!valid()) {
    glViewport(0,0,pixel_w(),pixel_h());
    glFrustum(...) or glOrtho(...)
    ...other initialization...
  }
  if (!context_valid()) {
    ...load textures, etc. ...
  }
  // clear screen
  glClearColor(...);
  glClear(...);
  ... draw your geometry here ...
}
```

Actual example code to clear screen to black and draw a 2D white "X":

```cpp
void mywindow::draw() {
  if (!valid()) {
    glLoadIdentity();
    glViewport(0,0,pixel_w(),pixel_h());
    glOrtho(-w(),w(),-h(),h(),-1,1);
  }
  // Clear screen
  glClearColor(GL_COLOR_BUFFER_BIT);
  // Draw white 'X'
  glColor3f(1.0, 1.0, 1.0);
  glBegin(GL_LINESPAIR); glVertex2f(w(), h()); glVertex2f(-w(), h()); glEnd();
  glBegin(GL_LINESPAIR); glVertex2f(w(), -h()); glVertex2f(-w(), -h()); glEnd();
}
```

Regular FLTK widgets can be added as children to the `Fl_Gl_Window`. To correctly overlay the widgets, `Fl_Gl_Window::draw()` must be called after rendering the main scene.

```cpp
void mywindow::draw() {
  // draw 3d graphics scene
  Fl_Gl_Window::draw();
  // -- or --
  draw_begin();
  Fl_Window::draw();
}
```
// other 2d drawing calls, overlays, etc.
draw_end();
}

Implements Fl_Widget.
Reimplemented in Fl_Glut_Window.

### 12.49.3.9 draw_begin()

```cpp
void Fl_Gl_Window::draw_begin ( ) [protected]
```

Supports drawing to an Fl_Gl_Window with the FLTK 2D drawing API.

See also

*Using FLTK widgets in OpenGL Windows*

### 12.49.3.10 draw_end()

```cpp
void Fl_Gl_Window::draw_end ( ) [protected]
```

To be used as a match for a previous call to Fl_Gl_Window::draw_begin().

See also

*Using FLTK widgets in OpenGL Windows*

### 12.49.3.11 flush()

```cpp
void Fl_Gl_Window::flush ( ) [virtual]
```

Forces the window to be drawn, this window is also made current and calls draw().
Reimplemented from Fl_Window.

### 12.49.3.12 handle()

```cpp
int Fl_Gl_Window::handle ( int event ) [virtual]
```

Handle some FLTK events as needed.
Reimplemented from Fl_Widget.
Reimplemented in Fl_Glut_Window.

### 12.49.3.13 hide()

```cpp
void Fl_Gl_Window::hide ( ) [virtual]
```

Hides the window and destroys the OpenGL context.
Reimplemented from Fl_Widget.

### 12.49.3.14 make_current()

```cpp
void Fl_Gl_Window::make_current ( )
```

The make_current() method selects the OpenGL context for the widget.
It is called automatically prior to the draw() method being called and can also be used to implement feedback and/or selection within the handle() method.

### 12.49.3.15 make_overlay_current()

```cpp
void Fl_Gl_Window::make_overlay_current ( )
```

Selects the OpenGL context for the widget's overlay.
This method is called automatically prior to the draw_overlay() method being called and can also be used to implement feedback and/or selection within the handle() method.
12.49.3.16  mode() [1/3]

Fl_Mode Fl_Gl_Window::mode ( ) const [inline]
Returns the current OpenGL capabilities of the window.
Don't use this if capabilities were set through Fl_Gl_Window::mode(const int *a).

12.49.3.17  mode() [2/3]

int Fl_Gl_Window::mode ( const int *a ) [inline]
Set the OpenGL capabilities of the window using platform-specific data.

Parameters

| a | zero-ending array of platform-specific attributes and attribute values |

Unix/Linux platform: attributes are GLX attributes adequate for the 3rd argument of the glXChooseVisual() function (e.g., GLX_DOUBLEBUFFER, defined by including <GL/glx.h>).

Note

What attributes are adequate here is subject to change. The preferred, stable public API is Fl_Gl_Window::mode(int a).

Windows platform: this member function is of no use.

Mac OS X platform: attributes belong to the CGLPixelFormatAttribute enumeration (defined by including <OpenGL/OpenGL.h>, e.g., kCGLPFADoubleBuffer) and may be followed by adequate attribute values.

12.49.3.18  mode() [3/3]

int Fl_Gl_Window::mode ( int a ) [inline]
Set or change the OpenGL capabilities of the window.
The value can be any of the following OR'd together:

- FL_RGB - RGB color (not indexed)
- FL_RGB8 - RGB color with at least 8 bits of each color
- FL_INDEX - Indexed mode
- FL_SINGLE - not double buffered
- FL_DOUBLE - double buffered
- FL_ACCUM - accumulation buffer
- FL_ALPHA - alpha channel in color
- FL_DEPTH - depth buffer
- FL_STENCIL - stencil buffer
- FL_MULTISAMPLE - multisample antialiasing
- FL_OPENGL3 - use OpenGL version 3.0 or more.

FL_RGB and FL_SINGLE have a value of zero, so they are "on" unless you give FL_INDEX or FL_DOUBLE.
If the desired combination cannot be done, FLTK will try turning off FL_MULTISAMPLE. If this also fails the show() will call Fl::error() and not show the window.

You can change the mode while the window is displayed. This is most useful for turning double-buffering on and off.
Under X this will cause the old X window to be destroyed and a new one to be created. If this is a top-level window this will unfortunately also cause the window to blink, raise to the top, and be de-iconized, and the xid() will change, possibly breaking other code. It is best to make the GL window a child of another window if you wish to do this!
mode() must not be called within draw() since it changes the current context.
The FL_OPENGL3 flag is required to access OpenGL version 3 or more under the X11 and MacOS platforms; it's optional under Windows and Wayland. See more details in Using OpenGL 3.0 (or higher versions).
12.49 Fl_Gl_Window Class Reference

12.49.3.19 ortho()

void Fl_Gl_Window::ortho ( )
Sets the projection so 0,0 is in the lower left of the window and each pixel is 1 unit wide/tall. If you are drawing 2D images, your draw() method may want to call this if valid() is false.

12.49.3.20 pixel_h()

int Fl_Gl_Window::pixel_h ( ) [inline]
Gives the window height in OpenGL pixels. When an Fl_Gl_Window is mapped to a HighDPI display, the value given by Fl_Gl_Window::h() which is expressed in FLTK units, may differ from the window height in pixels. Calls to OpenGL functions expecting pixel values (e.g., glViewport) should therefore use pixel_h() rather than h(). Method pixel_h() detects when the GUI is rescaled or when the window has been moved between low and high resolution displays and automatically adjusts the returned value.

Version 1.3.4

12.49.3.21 pixel_w()

int Fl_Gl_Window::pixel_w ( ) [inline]
Gives the window width in OpenGL pixels. When an Fl_Gl_Window is mapped to a HighDPI display, the value given by Fl_Gl_Window::w() which is expressed in FLTK units, may differ from the window width in pixels. Calls to OpenGL functions expecting pixel values (e.g., glViewport) should therefore use pixel_w() rather than w(). Method pixel_w() detects when the GUI is rescaled or when the window has been moved between low and high resolution displays and automatically adjusts the returned value.

Version 1.3.4

12.49.3.22 pixels_per_unit()

float Fl_Gl_Window::pixels_per_unit ( )
The number of pixels per FLTK unit of length for the window. This method dynamically adjusts its value when the GUI is rescaled or when the window is moved to/from displays of distinct resolutions. This method is useful, e.g., to convert, in a window's handle() method, the FLTK units returned by Fl::event_x() and Fl::event_y() to the pixel units used by the OpenGL source code.

Version 1.3.4

12.49.3.23 redraw_overlay()

void Fl_Gl_Window::redraw_overlay ( )
Causes draw_overlay() to be called at a later time. Initially the overlay is clear. If you want the window to display something in the overlay when it first appears, you must call this immediately after you show() your window.
12.49.3.24 resize()

void Fl_Gl_Window::resize (   
    int x,   
    int y,   
    int w,   
    int h ) [virtual]

Changes the size or position of the widget.
This is a virtual function so that the widget may implement its own handling of resizing. The default version does not call the redraw() method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position.

Some window managers under X11 call resize() a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations.

position(X, Y) is a shortcut for resize(X, Y, w(), h()), and size(W, H) is a shortcut for resize(x(), y(), W, H).

Parameters

| in | x, y | new position relative to the parent window |
| in | w, h | new size |

See also

position(int,int), size(int,int)

Reimplemented from Fl_Widget.

12.49.3.25 show()

void Fl_Gl_Window::show ( ) [virtual]

Makes a widget visible.
An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other events like FL_SHOW.
The visible() method returns true if the widget is set to be visible. The visible_r() method returns true if the widget and all of its parents are visible. A widget is only visible if visible() is true on it and all of its parents.
Changing it will send FL_SHOW or FL_HIDE events to the widget. Do not change it if the parent is not visible, as this will send false FL_SHOW or FL_HIDE events to the widget. redraw() is called if necessary on this or the parent.

See also

hide(), visible(), visible_r()

Reimplemented from Fl_Widget.

12.49.3.26 swap_buffers()

void Fl_Gl_Window::swap_buffers ( )
The swap_buffers() method swaps the back and front buffers.
It is called automatically after the draw() method is called.

12.49.3.27 swap_interval() [1/2]

int Fl_Gl_Window::swap_interval ( ) const

Gets the rate at which the GL windows swaps buffers.
This method can be called after the OpenGL context was created, typically within the user overridden Fl_Gl_Window::draw() method that will be overridden by the user.
Note

This method depends highly on the underlying OpenGL contexts and driver implementation. Some drivers return no information, most drivers don’t support intervals with multiple frames and return only 0 or 1. Some drivers have the ability to set the swap interval but no way to query it, hence this method may return -1 even though the interval was set correctly. Conversely a return value greater zero does not guarantee that the driver actually honors the setting.

Returns

an integer greater zero if vertical blanking is taken into account when swapping OpenGL buffers
0 if the vertical blanking is ignored
-1 if the information can not be retrieved

12.49.3.28 swap_interval() [2/2]

void Fl_Gl_Window::swap_interval (  
    int frames )

Sets the rate at which the GL windows swaps buffers.
This method can be called after the OpenGL context was created, typically within the user overridden Fl_Gl_Window::draw() method that will be overridden by the user.

Note

This method depends highly on the underlying OpenGL contexts and driver implementation. Most driver seem to accept only 0 and 1 to swap buffer asynchronously or in sync with the vertical blank.

Parameters

| in   | frames     | set the number of vertical frame blanks between OpenGL buffer swaps |

12.49.3.29 valid()

char Fl_Gl_Window::valid ( ) const [inline]

Is turned off when FLTK creates a new context for this window or when the window resizes, and is turned on after draw() is called.

You can use this inside your draw() method to avoid unnecessarily initializing the OpenGL context. Just do this:

```cpp
void mywindow::draw() { 
    if (!valid()) { 
        glViewport(0,0,pixel_w(),pixel_h());
        glFrustum(...);
        ...other initialization...
    }
    if (!context_valid()) { 
        ...load textures, etc. ...
    }
    ... draw your geometry here ... 
}
```

You can turn valid() on by calling valid(1). You should only do this after fixing the transformation inside a draw() or after make_current(). This is done automatically after draw() returns.

The documentation for this class was generated from the following files:

- Fl_Gl_Window.H
- Fl_Gl_Overlay.cxx
- Fl_Gl_Window.cxx

12.50 Fl_Glut_Bitmap_Font Struct Reference

fltk glut font/size attributes used in the glutXXX functions
#include <glut.H>

Generated by Doxygen
Public Attributes

• \texttt{Fl\_Font} font
• \texttt{Fl\_Fontsize} size

12.50.1 Detailed Description

\texttt{fltk} glut font/size attributes used in the glutXXX functions

The documentation for this struct was generated from the following file:

• glut.H

12.51 \texttt{Fl\_Glut\_StrokeChar} Struct Reference

Public Attributes

• int \texttt{Number}
• GLfloat \texttt{Right}
• const \texttt{Fl\_Glut\_StrokeStrip *} \texttt{Strips}

The documentation for this struct was generated from the following file:

• glut.H

12.52 \texttt{Fl\_Glut\_StrokeFont} Struct Reference

Public Attributes

• const \texttt{Fl\_Glut\_StrokeChar **} \texttt{Characters}
• GLfloat \texttt{Height}
• char * \texttt{Name}
• int \texttt{Quantity}

The documentation for this struct was generated from the following file:

• glut.H

12.53 \texttt{Fl\_Glut\_StrokeStrip} Struct Reference

Public Attributes

• int \texttt{Number}
• const \texttt{Fl\_Glut\_StrokeVertex *} \texttt{Vertices}

The documentation for this struct was generated from the following file:

• glut.H

12.54 \texttt{Fl\_Glut\_StrokeVertex} Struct Reference

Public Attributes

• GLfloat \texttt{X}
• GLfloat \texttt{Y}

The documentation for this struct was generated from the following file:

• glut.H
GLUT is emulated using this window class and these static variables (plus several more static variables hidden in `glut_compatibility.cxx`):

```cpp
#include <glut.H>
```

Inheritance diagram for Fl_Glut_Window:

```
Fl_Widget
    ↓
Fl_Group
    ↓
Fl_Window
    ↓
Fl_Gl_Window
    ↓
Fl_Glut_Window
```

Public Member Functions

- **Fl_Glut_Window** (int w, int h, const char *t=0)
  
  Creates a glut window, registers to the glut windows list.

- **Fl_Glut_Window** (int x, int y, int w, int h, const char *t=0)
  
  Creates a glut window, registers to the glut windows list.

- `void make_current()`

- `~Fl_Glut_Window()`
  
  Destroys the glut window, first unregister it from the glut windows list.

Public Member Functions inherited from Fl_Gl_Window

- `Fl_Gl_Window const * as_gl_window () const FL_OVERRIDE`

- `Fl_Gl_Window * as_gl_window () FL_OVERRIDE`
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- `int can_do ()`
  
  Returns non-zero if the hardware supports the current OpenGL mode.

- `int can_do_overlay ()`
  
  Returns true if the hardware overlay is possible.

- `GLContext context () const`
  
  Returns a pointer to the window's OpenGL rendering context.

- `void context (GLContext, int destroy_flag=0)`
  
  Sets a pointer to the GLContext that this window is using.

- `char context_valid () const`
  
  Will only be set if the OpenGL context is created or recreated.

- `void context_valid (char v)`
  
  See char Fl_Gl_Window::context_valid() const.

- **Fl_Gl_Window** (int W, int H, const char *l=0)
  
  Creates a new Fl_Gl_Window widget using the given size, and label string.

- **Fl_Gl_Window** (int X, int Y, int W, int H, const char *l=0)
  
  Creates a new Fl_Gl_Window widget using the given position, size, and label string.

- `void flush () FL_OVERRIDE`
  
  Forces the window to be drawn, this window is also made current and calls draw().

- `void hide () FL_OVERRIDEE`
Hides the window and destroys the OpenGL context.

- **void hide_overlay()**
  - Hides the window if it is not this window, does nothing in Windows.

- **void invalidate()**
  - The invalidate() method turns off valid() and is equivalent to calling value(0).

- **void make_current()**
  - The make_current() method selects the OpenGL context for the widget.

- **void make_overlay_current()**
  - Selects the OpenGL context for the widget’s overlay.

- **FL_Mode mode() const**
  - Returns the current OpenGL capabilities of the window.

- **int mode(const int*a)**
  - Set the OpenGL capabilities of the window using platform-specific data.

- **int mode(int a)**
  - Set or change the OpenGL capabilities of the window.

- **void ortho()**
  - Sets the projection so 0,0 is in the lower left of the window and each pixel is 1 unit wide/tall.

- **int pixel_h()**
  - Gives the window height in OpenGL pixels.

- **int pixel_w()**
  - Gives the window width in OpenGL pixels.

- **float pixels_per_unit()**
  - The number of pixels per FLTK unit of length for the window.

- **void redraw_overlay()**
  - Causes draw_overlay() to be called at a later time.

- **void resize(int, int, int, int) FL_OVERRIDE**
  - Changes the size or position of the widget.

- **void show() FL_OVERRIDE**
  - Makes a widget visible.

- **void show(int a, char**∗∗∗∗b)**
  - Same as Fl_Window::show(int a, char**∗∗b)

- **void swapBuffers()**
  - The swap_buffers() method swaps the back and front buffers.

- **int swap_interval() const**
  - Gets the rate at which the GL windows swaps buffers.

- **void swap_interval(int)**
  - Sets the rate at which the GL windows swaps buffers.

- **char valid() const**
  - Is turned off when FLTK creates a new context for this window or when the window resizes, and is turned on after draw() is called.

- **void valid(char v)**
  - See char Fl_Gl_Window::valid() const.

- **~Fl_Gl_Window()**
  - The destructor removes the widget and destroys the OpenGL context associated with it.
Public Member Functions inherited from Fl_Window

- virtual class Fl_Double_Window * as_double_window ()
  Return non-null if this is an Fl_Double_Window object.
- virtual class Fl_Overlay_Window * as_overlay_window ()
  Return non-null if this is an Fl_Overlay_Window object.
- Fl_Window const * as_window () const FL_OVERRIDE
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int border () const
  Returns whether the window possesses a border.
- void border (int b)
  Sets whether or not the window manager border is around the window.
- void clear_border ()
  Fast inline function to turn the window manager border off.
- void clear_modal_states ()
  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.
- void copy_label (const char *a)
  Sets the window titlebar label to a copy of a character string.
- void cursor (const Fl_RGB_Image *c, int, int)
  Changes the cursor for this window using the provided image as cursor's shape.
- void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  For back compatibility only.
- void cursor (Fl_Cursor)
  Changes the cursor for this window.
- int decorated_h () const
  Returns the window height including any window title bar and any frame added by the window manager.
- int decorated_w () const
  Returns the window width including any frame added by the window manager.
- void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  For back compatibility only.
- void default_cursor (Fl_Cursor)
  Sets the default window cursor.
- void draw_backdrop ()
  Draw the background image if one is set and is aligned inside.
- Fl_Window (int w, int h, const char *title=0)
  Creates a window from the given width w, height h, and title.
- Fl_Window (int x, int y, int w, int h, const char *title=0)
  Creates a window from the given position (x, y), size (w, h) and title.
- void free_position ()
  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.
- void fullscreen ()
  Makes the window completely fill one or more screens, without any window manager border visible.
- unsigned int fullscreen_active () const
  Returns non zero if FULLSCREEN flag is set, 0 otherwise.
- void fullscreen_off ()
  Turns off any side effects of fullscreen()
- void fullscreen_off (int X, int Y, int W, int H)
  Turns off any side effects of fullscreen() and does resize(x,y,w,h).
- void fullscreen_screens (int top, int bottom, int left, int right)
Sets which screens should be used when this window is in fullscreen mode.

- int handle (int) FL_OVERRIDE
  Handles the specified event.

- void hide () FL_OVERRIDE
  Removes the window from the screen.

- void hotspot (const Fl_Widget &p, int offscreen=0)
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

- void hotspot (const Fl_Widget *, int offscreen=0)
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

- void hotspot (int x, int y, int offscreen=0)
  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.

- const void * icon () const
  Gets the current icon window target dependent data.

- void icon (const Fl_RGB_Image *)
  Sets or resets a single window icon.

- void icon (const void *ic)
  Platform-specific method to set the window icon usable on Windows and X11 only.

- void iconize ()
  Iconifies the window.

- const char * iconlabel () const
  See void Fl_Window::iconlabel(const char *)

- void iconlabel (const char *)
  Sets the icon label.

- void icons (const Fl_RGB_Image *[ ], int)
  Sets the window icons.

- void icons (HICON big_icon, HICON small_icon)
  Sets the window icons using HICON handles (Windows platform only).

- const char * label () const
  See void Fl_Window::label(const char *)

- void label (const char *)
  Sets the window title bar label.

- void label (const char *label, const char *iconlabel)
  Sets the icon label.

- void make_current ()
  Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.

- void maximize ()
  Maximizes a top-level window to its current screen.

- unsigned int maximize_active () const
  Returns whether the window is currently maximized.

- unsigned int menu_window () const
  Returns true if this window is a menu window.

- unsigned int modal () const
  Returns true if this window is modal.

- unsigned int non_modal () const
  Returns true if this window is modal or non-modal.

- fl_uintptr_t os_id ()
  Returns a platform-specific identification of a shown window, or 0 if not shown.

- unsigned int override () const
Returns non zero if OVERRIDE flag is set, 0 otherwise.

\- void \texttt{resize} (int X, int Y, int W, int H) \texttt{FL\_OVERRIDE}

Changes the size and position of the window.

\- int \texttt{screen\_num} ()

The number of the screen containing the mapped window.

\- void \texttt{screen\_num} (int screen\_num)

Set the number of the screen where to map the window.

\- void \texttt{set\_menu\_window} ()

Marks the window as a menu window.

\- void \texttt{set\_modal} ()

A ”modal” window, when \texttt{shown()}, will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the ”transient for” property).

\- void \texttt{set\_non\_modal} ()

A ”non-modal” window (terminology borrowed from Microsoft Windows) acts like a \texttt{modal()} one in that it remains on top, but it has no effect on event delivery.

\- void \texttt{set\_override} ()

Activates the flags NOBORDER/OVERRIDE.

\- void \texttt{set\_tooltip\_window} ()

Marks the window as a tooltip window.

\- const \texttt{Fl\_Image} \* \texttt{shape} ()

Returns the image controlling the window shape or NULL.

\- void \texttt{shape} (const \texttt{Fl\_Image} \&b)

Set the window's shape with an \texttt{Fl\_Image}.

\- void \texttt{shape} (const \texttt{Fl\_Image} \*img)

Assigns a non-rectangular shape to the window.

\- void \texttt{show} () \texttt{FL\_OVERRIDE}

Puts the window on the screen.

\- void \texttt{show} (int argc, char \*=\*argv)

Puts the window on the screen with \texttt{show()} and parses command-line arguments.

\- int \texttt{shown} ()

Returns non-zero if \texttt{show()} has been called (but not \texttt{hide()}).

\- void \texttt{size\_range} (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)

Sets the allowable range the user can resize this window to.

\- unsigned int \texttt{tooltip\_window} () const

Returns true if this window is a tooltip window.

\- void \texttt{un\_maximize} ()

Returns a previously maximized top-level window to its previous size.

\- void \texttt{wait\_for\_expose} ()

Waits for the window to be displayed after calling \texttt{show()}.

\- int \texttt{x\_root} () const

Gets the x position of the window on the screen.

\- const char \* \texttt{xclass} () const

Returns the xclass for this window, or a default.

\- void \texttt{xclass} (const char \*=c)

Sets the xclass for this window.

\- int \texttt{y\_root} () const

Gets the y position of the window on the screen.

\- virtual \texttt{~FL\_Window} ()

The destructor also deletes all the children.
Public Member Functions inherited from Fl_Group

- **Fl_Widget** *& _ddfdesign_kludge ()
  This is for forms compatibility only.

- **void add (Fl_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.

- **void add (Fl_Widget &o)
  See void Fl_Group::add(Fl_Widget &w)

- **void add_resizable (Fl_Widget &o)
  Adds a widget to the group and makes it the resizable widget.

- **Fl_Widget** *const & array () const
  Returns a pointer to the array of children.

- **Fl_Group** *const as_group () const FL_OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- **void begin ()
  Sets the current group so you can build the widget tree by just constructing the widgets.

- **Fl_Widget** *child (int n) const
  Returns array()[n].

- **int children () const
  Returns how many child widgets the group has.

- **void clear ()
  Deletes all child widgets from memory recursively.

- **unsigned int clip_children ()
  Returns the current clipping mode.

- **void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- **virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.

- **void end ()
  Exactly the same as current(this->parent()).

- **int find (const Fl_Widget &o) const
  See int Fl_Group::find(const Fl_Widget &w) const.

- **int find (const Fl_Widget &)
  Searches the child array for the widget and returns the index.

- **Fl_Group** (int, int, int, int, const char * = 0)
  Creates a new Fl_Group widget using the given position, size, and label string.

- **void focus (Fl_Widget &W)

- void forms_end ()
  This is for forms compatibility only.

- **void init_sizes ()
  Resets the internal array of widget sizes and positions.

- **void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.

- **void insert (Fl_Widget &o, Fl_Widget *before)
  This does insert(w, find(before)).

- **void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.

- **void remove (Fl_Widget &o)
  Removes the widget o from the group.

- **void remove (int index)
Removes the widget at index from the group but does not delete it.

- \textbf{Fl_Widget *} resizable () const
  
  Returns the group's resizable widget.

- void resizable (Fl_Widget &o)
  
  Sets the group's resizable widget.

- void resizable (Fl_Widget *o)
  
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- virtual ~Fl_Group ()
  
  The destructor also deletes all the children.

**Public Member Functions inherited from Fl_Widget**

- void \_clear_fullscreen ()
- void \_set_fullscreen ()
- void activate ()
  
  Activates the widget.

- unsigned int active () const
  
  Returns whether the widget is active.

- int active_r () const
  
  Returns whether the widget and all of its parents are active.

- Fl_Align align () const
  
  Gets the label alignment.

- void align (Fl_Align alignment)
  
  Sets the label alignment.

- long argument () const
  
  Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)
  
  Sets the current user data (long) argument that is passed to the callback function.

- void bind_deimage (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage (int f)
  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)
  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- Fl_Boxtype box () const
  
  Gets the box type of the widget.

- void box (Fl_Boxtype new_box)
  
  Sets the box type for the widget.

- Fl_Callback_p callback () const
  
  Gets the current callback function for the widget.

- void callback (Fl_Callback *cb)
  
  Sets the current callback function for the widget.

- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  
  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback *cb, void *p)
  
  Sets the current callback function and data for the widget.

- void callback (Fl_Callback0 *cb)
  
  Sets the current callback function for the widget.

- void callback (Fl_Callback1 *cb, long p=0)
Sets the current callback function for the widget.

- unsigned int changed () const
  Checks if the widget value changed since the last callback.
- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()
  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)
  Clears or sets the damage flags.
- void clear_output ()
  Sets a widget to accept input.
- void clear_visible ()
  Hides the widget.
- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
- FL_Color color () const
  Gets the background color of the widget.
- void color (FL_Color bg)
  Sets the background color of the widget.
- void color (FL_Color bg, FL_Color sel)
  Sets the background and selection color of the widget.
- FL_Color color2 () const
  For back compatibility only.
- void color2 (unsigned a)
  For back compatibility only.
- int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.
- void copy_label (const char *new_label)
  Sets the current label.
- void copy_tooltip (const char *text)
  Sets the current tooltip text.
- uchar damage () const
  Returns non-zero if draw() needs to be called.
- void damage (uchar c)
  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
- int damage_resize (int, int, int, int)
  Internal use only.
- void deactivate ()
  Deactivates the widget.
- FL_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
- const FL_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (FL_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (FL_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (FL_Callback Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (FL_Widget *widget, long arg, FL_Callback Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (FL_Widget *widget, void *arg=0, FL_Callback Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, FL_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• FL_Image *image ()
  Gets the image that is used as part of the widget label when in the active state.
• const FL_Image *image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (FL_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (FL_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const FL_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char *label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (FL_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• FL_Color labelcolor () const
  Gets the label color.
• void labelcolor (FL_Color c)
  Sets the label color.
• FL_Font labelfont () const
  Gets the font to use.
• void labelfont (FL_Font f)
  Sets the font to use.
• FL_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (FL_Fontsize pix)
  Sets the font size in pixels.
• FL_Labeltype labeltype () const
  Gets the label type.
• void labeltype (FL_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
Sets whether this widget needs a keyboard.

- **unsigned int output () const**
  
  Returns if a widget is used for output only.

- **Fl_Group * parent () const**
  
  Returns a pointer to the parent widget.

- **void parent (Fl_Group *p)**
  
  Internal use only - "for hacks only".

- **void position (int X, int Y)**
  
  Repositions the window or widget.

- **void redraw ()**
  
  Schedules the drawing of the widget.

- **void redraw_label ()**
  
  Schedules the drawing of the label.

- **Fl_Color selection_color () const**
  
  Gets the selection color.

- **void selection_color (Fl_Color a)**
  
  Sets the selection color.

- **void set_active ()**
  
  Marks the widget as active without sending events or changing focus.

- **void set_changed ()**
  
  Marks the value of the widget as changed.

- **void set_output ()**
  
  Sets a widget to output only.

- **void set_visible ()**
  
  Makes the widget visible.

- **void set_visible_focus ()**
  
  Enables keyboard focus navigation with this widget.

- **int shortcut_label () const**
  
  Returns whether the widget's label uses '&' to indicate shortcuts.

- **void shortcut_label (int value)**
  
  Sets whether the widget's label uses '&' to indicate shortcuts.

- **void size (int W, int H)**
  
  Changes the size of the widget.

- **int take_focus ()**
  
  Gives the widget the keyboard focus.

- **unsigned int takesevents () const**
  
  Returns if the widget is able to take events.

- **int testShortcut ()**
  
  Returns true if the widget's label contains the entered 'x' shortcut.

- **const char * tooltip () const**
  
  Gets the current tooltip text.

- **void tooltip (const char *text)**
  
  Sets the current tooltip text.

- **Fl_Window * top_window () const**
  
  Returns a pointer to the top-level window for the widget.

- **Fl_Window * top_window_offset (int &xoff, int &yoff) const**
  
  Finds the x/y offset of the current widget relative to the top-level window.

- **uchar type () const**
  
  Gets the widget type.

- **void type (uchar t)**
  
  Sets the widget type.
• int **use_accents_menu** ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void * user_data () const
  Gets the user data for this widget.

• void **user_data** (**Fl_Callback_User_Data** ∗v, bool auto_free)
  Sets the user data for this widget.

• void **user_data** (void ∗v)
  Sets the user data for this widget.

• unsigned int visible () const
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

• void **visible_focus** (int v)
  Modifies keyboard focus navigation.

• int visible_r () const
  Returns whether a widget and all its parents are visible.

• int **w** () const
  Gets the widget width.

• **Fl_When** when () const
  Returns the conditions under which the callback is called.

• void **when** (uchar i)
  Sets the flags used to decide when a callback is called.

• **Fl_Window** ∗ window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int **x** () const
  Gets the widget position in its window.

• int **y** () const
  Gets the widget position in its window.

• virtual ∼**Fl_Widget** ()
  Destroys the widget.

Public Attributes

• void(* display *)()
• void(* entry *) (int)
• void(* keyboard *) (uchar, int x, int y)
• int menu [3]
• void(* motion *) (int x, int y)
• void(* mouse *) (int b, int state, int x, int y)
• int number
• void(* overlaydisplay *)()
• void(* passivemotion *) (int x, int y)
• void(* reshape *) (int w, int h)
• void(* special *) (int, int x, int y)
• void(* visibility *) (int)

Protected Member Functions

• void draw () FL_OVERRIDEdraws the FI GI_Window.
• void draw_overlay () FL_OVERRIDE
  You must implement this virtual function if you want to draw into the overlay.
• int handle (int) FL_OVERRIDE
  Handle some FLTK events as needed.
Protected Member Functions inherited from `Fl_Gl_Window`

- `void draw_begin ()`
  Supports drawing to an `Fl_Gl_Window` with the FLTK 2D drawing API.
- `void draw_end ()`
  To be used as a match for a previous call to `Fl_Gl_Window::draw_begin()`.

Protected Member Functions inherited from `Fl_Window`

- `void default_size_range ()`
  Protected method to calculate the default size range of a window.
- `void draw () FL_OVERRIDE`
  Draws the widget.
- `int force_position () const`
  Returns the internal state of the window's FORCE_POSITION flag.
- `void force_position (int force)`
  Sets an internal flag that tells FLTK and the window manager to honor position requests.
- `void free_icons ()`
  Deletes all icons previously attached to the window.
- `int is_resizable ()`
  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from `Fl_Group`

- `Fl_Rect * bounds ()`
  Returns the internal array of widget sizes and positions.
- `void draw_child (Fl_Widget &widget) const`
  Forces a child to redraw.
- `void draw_children ()`
  Draws all children of the group.
- `void draw_outside_label (const Fl_Widget &widget) const`
  Parents normally call this to draw outside labels of child widgets.
- `virtual int on_insert (Fl_Widget *, int)`
  Allow derived groups to act when a widget is added as a child.
- `virtual int on_move (int, int)`
  Allow derived groups to act when a widget is moved within the group.
- `virtual void on_remove (int)`
  Allow derived groups to act when a child widget is removed from the group.
- `int * sizes ()`
  Returns the internal array of widget sizes and positions.
- `void update_child (Fl_Widget &widget) const`
  Draws a child only if it needs it.

Protected Member Functions inherited from `Fl_Widget`

- `void clear_flag (unsigned int c)`
  Clears a flag in the flags mask.
- `void draw_backdrop () const`
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- `void draw_box () const`
  Draws the widget box according its box style.
- `void draw_box (Fl_Boxtype t, Fl_Color c) const`
Draws a box of type t, of color c at the widget's position and size.

- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- void draw_focus () const
  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h) const
  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

- void draw_label () const
  Draws the widget's label at the defined label position.

- void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.

- unsigned int flags () const
  Gets the widget flags mask.

- void h (int v)
  Internal use only.

- void set_flag (unsigned int c)
  Sets a flag in the flags mask.

- void w (int v)
  Internal use only.

- void x (int v)
  Internal use only.

- void y (int v)
  Internal use only.

Additional Inherited Members

Public Types inherited from Fl_Window

- typedef struct HICON__ * HICON

Static Public Member Functions inherited from Fl_Gl_Window

- static int can_do (const int *m)
  Returns non-zero if the hardware supports the given OpenGL mode.

- static int can_do (int m)
  Returns non-zero if the hardware supports the given OpenGL mode.

Static Public Member Functions inherited from Fl_Window

- static Fl_Window * current ()
  Returns the last window that was made current.

- static void default_callback (Fl_Window *, void *)
  Back compatibility: Sets the default callback v for win to call on close event.

- static void default_icon (const Fl_RGB_Image *)
  Sets a single default window icon.

- static void default_icons (const Fl_RGB_Image *, int)
  Sets the default window icons.

- static void default_icons (HICON big_icon, HICON small_icon)
  Sets the default window icons (Windows platform only).
- static const char * default_xclass ()
  Returns the default xclass.
- static void default_xclass (const char *)
  Sets the default window xclass.
- static bool is_a_rescale ()
  Returns true when a window is being rescaled.
- static char show_next_window_iconic ()
  Returns the static flag whether the next window should be opened iconified.
- static void show_next_window_iconic (char stat)
  Sets a static flag whether the next window should be opened iconified.

**Static Public Member Functions inherited from Fl_Group**

- static Fl_Group * current ()
  Returns the currently active group.
- static void current (Fl_Group *g)
  Sets the current group.

**Static Public Member Functions inherited from Fl_Widget**

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don’t set a callback.
- static unsigned int label_shortcut (const char *)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

**Protected Types inherited from Fl_Widget**

```c
enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
```

flags possible values enumeration.

**Static Protected Attributes inherited from Fl_Window**

- static Fl_Window * current_
  Stores the last window that was made current.

### 12.55.1 Detailed Description

GLUT is emulated using this window class and these static variables (plus several more static variables hidden in glut_compatibility.cxx):
12.55.2 Member Function Documentation

12.55.2.1 draw()

void Fl_Glut_Window::draw ()
    { [protected], [virtual]

Draws the Fl_Gl_Window.
You must subclass Fl_Gl_Window and provide an implementation for draw(). You may also provide an implementation of draw_overlay() if you want to draw into the overlay planes. You can avoid reinitializing the viewport and lights and other things by checking valid() at the start of draw() and only doing the initialization if it is false.
The draw() method can only use OpenGL calls. Do not attempt to call X, any of the functions in <FL/fl_draw.H>,
or glX directly. Do not call gl_start() or gl_finish().
If double-buffering is enabled in the window, the back and front buffers are swapped after this function is completed.
The following pseudo-code shows how to use "if (!valid())" to initialize the viewport:

```cpp
void mywindow::draw() {
    if (!valid()) {
        glViewport(0,0,pixel_w(),pixel_h());
        glFrustum(...) or glOrtho(...)
        ...other initialization ...
    if (!context_valid()) {
        ...load textures, etc. ...
    }
    // clear screen
    glClearColor(...);
    glClear(...);
    // draw your geometry here ...
}
```

Actual example code to clear screen to black and draw a 2D white "X":

```cpp
void mywindow::draw() {
    if (!valid()) {
        glLoadIdentity();
        glViewport(0,0,pixel_w(),pixel_h());
        glOrtho(-w(),w(),-h(),h(),-1,1);
    }
    // Clear screen
    glClearColor(GL_COLOR_BUFFER_BIT);
    // Draw white 'X'
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINES); glVertex2f(w(), h()); glVertex2f(-w(),-h()); glEnd();
    glBegin(GL_LINES); glVertex2f(w(),-h()); glVertex2f(-w(), h()); glEnd();
}
```

Regular FLTK widgets can be added as children to the Fl_Gl_Window. To correctly overlay the widgets, Fl_Gl_Window::draw() must be called after rendering the main scene.

```cpp
void mywindow::draw() {
    // draw 3d graphics scene
    Fl_Gl_Window::draw();
    // -- or --
    draw_begin();
    Fl_Window::draw();
    // other 2d drawing calls, overlays, etc.
    draw_end();
}
```

Reimplemented from Fl_Gl_Window.

12.55.2.2 draw_overlay()

void Fl_Glut_Window::draw_overlay ( )
    { [protected], [virtual]

You must implement this virtual function if you want to draw into the overlay.
The overlay is cleared before this is called. You should draw anything that is not clear using OpenGL. You must use
gl_color(i) to choose colors (it allocates them from the colormap using system-specific calls), and remember that you are in an indexed OpenGL mode and drawing anything other than flat-shaded will probably not work.
Both this function and Fl_Gl_Window::draw() should check Fl_Gl_Window::valid() and set the same transformation.
If you don't your code may not work on other systems. Depending on the OS, and on whether overlays are real or simulated, the OpenGL context may be the same or different between the overlay and main window.
Reimplemented from Fl_Gl_Window.

12.55.2.3 handle()

int Fl_Glut_Window::handle ( int event )
    { [protected], [virtual]
Handle some FLTK events as needed. 
Reimplemented from Fl_Gl_Window. 
The documentation for this class was generated from the following files:

- glut.H
- glut_compatibility.cxx

12.56  Fl_Grid Class Reference

Fl_Grid is a container (layout) widget with multiple columns and rows.
#include <Fl_Grid.H>
Inheritance diagram for Fl_Grid:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Grid
```

Classes

- class Cell

Public Member Functions

- Fl_Grid::Cell * cell (Fl_Widget * widget) const
  Get the grid cell of widget widget.
- Fl_Grid::Cell * cell (int row, int col) const
  Get the grid cell of row row and column col.
- virtual void clear_layout ()
  Reset the layout w/o removing widgets.
- void col_gap (const int * value, size_t size)
  Set more than one column gaps at once.
- int col_gap (int col) const
  Set the gap of column col.
- void col_gap (int col, int value)
  Set the gap of column col.
- void col_weight (const int * value, size_t size)
  Set the weight of more than one column.
- int col_weight (int col) const
  Set the weight of a column.
- void col_width (const int * value, size_t size)
  Set minimal widths of more than one column.
- int col_width (int col) const
  Set the minimal width of a column.
- short cols () const
- int computed_col_width (int col) const
- int computed_row_height (int row) const
- void debug (int level=127)
  Output layout information of this Fl_Grid to stderr.
- **Fl_Grid** (int X, int Y, int W, int H, const char *L=0)
  
  Create a new Fl_Grid widget.

- void **gap** (int *row_gap, int *col_gap) const
  
  Get the default gaps for rows and columns.

- virtual void **gap** (int row_gap, int col_gap=-1)
  
  Set default gaps for rows and columns.

- virtual void **layout** ()
  
  Calculate the grid layout and resize and position all widgets.

- virtual void **layout** (int rows, int cols, int margin=-1, int gap=-1)
  
  Set the basic layout parameters of the Fl_Grid widget.

- int **margin** (int *left, int *top, int *right, int *bottom) const
  
  Returns all outside margin sizes of the grid.

- virtual void **margin** (int left, int top=-1, int right=-1, int bottom=-1)
  
  Set all margins (left, top, right, bottom).

- bool **need_layout** () const
  
  Return whether layout calculation is required.

- void **need_layout** (int set)
  
  Request or reset the request to calculate the layout of children.

- virtual void **resize** (int X, int Y, int W, int H) FL_OVERRIDE
  
  Recalculate the layout and position and resize all widgets.

- int **row_gap** (int row) const
  
  Set the gap of row row.

- void **row_gap** (const int *value, size_t size)
  
  Set more than one row gaps at once.

- int **row_gap** (int row) const
  
  Set the gap of row row.

- void **row_gap** (int row, int value)
  
  Set the gap of row row.

- int **row_height** (int row) const
  
  Set the minimal row height of more than one row.

- void **row_height** (int row, int value)
  
  Set the minimal row height of row row.

- int **row_weight** (int row) const
  
  Set the weight of more than one row.

- void **row_weight** (int row, int value)
  
  Set the row weight of row row.

- short **rows** () const
  
  Enable or disable drawing of the grid helper lines for visualization.

- void **show_grid** (int set)
  
  Enable or disable drawing of the grid helper lines for visualization.

- **Fl_Grid::Cell** *widget** (Fl_Widget *wi, int row, int col, Fl_Grid_Align align=FL_GRID_FILL)
  
  Assign a widget to a grid cell and set its alignment.

- **Fl_Grid::Cell** *widget** (Fl_Widget *wi, int row, int col, int rowspan, int colspan, Fl_Grid_Align align=FL_GRID_FILL)
  
  Assign a widget to a grid cell and set cell spanning and alignment.
Public Member Functions inherited from Fl_Group

- **Fl_Widget** & _ddfdesign_kludge ()
  
  This is for forms compatibility only.

- void **add** (Fl_Widget &)
  
  The widget is removed from its current group (if any) and then added to the end of this group.

- void **add** (Fl_Widget *)
  
  See void Fl_Group::add(Fl_Widget &w)

- void **add_resizable** (Fl_Widget &o)
  
  Adds a widget to the group and makes it the resizable widget.

- **Fl_Widget** *const * array () const
  
  Returns a pointer to the array of children.

- **Fl_Group** *const * as_group () const FL_OVERRIDE

  Returns an Fl_Group pointer if this widget is an Fl_Group.

- void **begin** ()
  
  Sets the current group so you can build the widget tree by just constructing the widgets.

- **Fl_Widget** * child (int n) const
  
  Returns array()[n].

- int **children** () const
  
  Returns how many child widgets the group has.

- void **clear** ()
  
  Deletes all child widgets from memory recursively.

- unsigned int **clip_children** ()
  
  Returns the current clipping mode.

- void **clip_children** (int c)
  
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- virtual int **delete_child** (int n)
  
  Removes the widget at index from the group and deletes it.

- void **end** ()
  
  Exactly the same as current(this->parent()).

- int **find** (const Fl_Widget &o) const
  
  See int Fl_Group::find(const Fl_Widget *w) const.

- int **find** (const Fl_Widget *) const
  
  Searches the child array for the widget and returns the index.

- **Fl_Group** (int, int, int, int, const char * = 0)
  
  Creates a new Fl_Group widget using the given position, size, and label string.

- void **focus** (Fl_Widget *)

- void **forms_end** ()
  
  This is for forms compatibility only.

- int **handle** (int) FL_OVERRIDE
  
  Handles the specified event.

- void **init_sizes** ()
  
  Resets the internal array of widget sizes and positions.

- void **insert** (Fl_Widget &, int i)
  
  The widget is removed from its current group (if any) and then inserted into this group.

- void **insert** (Fl_Widget &, Fl_Widget *)
  
  This does insert(w, find(before)).

- void **remove** (Fl_Widget &)
  
  Removes a widget from the group but does not delete it.

- void **remove** (Fl_Widget *)

Generated by Doxygen
Removes the widget from the group.

- void `remove` (int index)
  Removes the widget at index from the group but does not delete it.

- Fl_Widget * `resizable` () const
  Returns the group’s resizable widget.

- void `resizable` (Fl_Widget &o)
  Sets the group’s resizable widget.

- void `resizable` (Fl_Widget *o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- void `resize` (int, int, int, int) FL_OVERRIDE
  Resizes the Fl_Group widget and all of its children.

- virtual ~Fl_Group ()
  The destructor also deletes all the children.

### Public Member Functions inherited from Fl_Widget

- void `_clear_fullscreen` ()
- void `_set_fullscreen` ()
- void `activate` ()
  Activates the widget.

- unsigned int `active` () const
  Returns whether the widget is active.

- int `active_r` () const
  Returns whether the widget and all of its parents are active.

- Fl_Align `align` () const
  Gets the label alignment.

- void `align` (Fl_Align alignment)
  Sets the label alignment.

- long `argument` () const
  Gets the current user data (long) argument that is passed to the callback function.

- void `argument` (long v)
  Sets the current user data (long) argument that is passed to the callback function.

- virtual class Fl_Gl_Window * `as_gl_window` ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const * `as_gl_window` () const
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual Fl_Window * `as_window` ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- void `bind_deimage` (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

- void `bind_deimage` (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void `bind_image` (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.

- void `bind_image` (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- Fl_Boxtype `box` () const
  Gets the box type of the widget.

- void `box` (Fl_Boxtype new_box)
  Sets the box type for the widget.

- Fl_Callback_p `callback` () const
Gets the current callback function for the widget.

• void callback (Fl_Callback ∗cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

• void callback (Fl_Callback ∗cb, void ∗p)
  Sets the current callback function and data for the widget.

• void callback (Fl_Callback0 ∗cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback1 ∗cb, long p=0)
  Sets the current callback function for the widget.

• unsigned int changed () const
  Checks if the widget value changed since the last callback.

• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

• void clear_changed ()
  Marks the value of the widget as unchanged.

• void clear_damage (uchar c=0)
  Clears or sets the damage flags.

• void clear_output ()
  Sets a widget to accept input.

• void clear_visible ()
  Hides the widget.

• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

• Fl_Color color () const
  Gets the background color of the widget.

• void color (Fl_Color bg)
  Sets the background color of the widget.

• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

• Fl_Color color2 () const
  For back compatibility only.

• void color2 (unsigned a)
  For back compatibility only.

• int contains (const Fl_Widget ∗w) const
  Checks if w is a child of this widget.

• void copy_label (const char ∗new_label)
  Sets the current label.

• void copy_tooltip (const char ∗text)
  Sets the current tooltip text.

• uchar damage () const
  Returns non-zero if draw() needs to be called.

• void damage (uchar c)
  Sets the damage bits for the widget.

• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

• int damage_resize (int, int, int, int)
  Internal use only.

• void deactivate ()
  Deactivates the widget.
- `Fl_Image * deimage ()`
  Gets the image that is used as part of the widget label when in the inactive state.

- `const Fl_Image * deimage () const`
  Gets the image that is used as part of the widget label when in the inactive state.

- `void deimage (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the inactive state.

- `void deimage (Fl_Image *img)`
  Sets the image to use as part of the widget label when in the inactive state.

- `int deimage_bound () const`
  Returns whether the inactive image is managed by the widget.

- `void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with default arguments.

- `void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.

- `void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.

- `void draw_label (int, int, int, int, Fl_Align) const`
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- `int h () const`
  Gets the widget height.

- `virtual void hide ()`
  Makes a widget invisible.

- `Fl_Image * image ()`
  Gets the image that is used as part of the widget label when in the active state.

- `const Fl_Image * image () const`
  Gets the image that is used as part of the widget label when in the active state.

- `void image (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the active state.

- `void image (Fl_Image *img)`
  Sets the image to use as part of the widget label when in the active state.

- `int image_bound () const`
  Returns whether the image is managed by the widget.

- `int inside (const Fl_Widget *wgt) const`
  Checks if this widget is a child of `wgt`.

- `int is_label_copied () const`
  Returns whether the current label was assigned with `copy_label()`.

- `const char * label () const`
  Gets the current label text.

- `void label (const char *text)`
  Sets the current label pointer.

- `void label (Fl_Labeltype a, const char *b)`
  Shortcut to set the label text and type in one call.

- `Fl_Color labelcolor () const`
  Gets the label color.

- `void labelcolor (Fl_Color c)`
  Sets the label color.

- `Fl_Font labelfont () const`
  Gets the font to use.

- `void labelfont (Fl_Font f)`
  Sets the font to use.

- `Fl_Fontsize labelsize () const`
**Class Documentation**

- void `labelsize (Fl_Fontsize pix)`  
  Gets the font size in pixels.

- void `labeltype (Fl_Labeltype a)`  
  Sets the label type.

- void `measure_label (int &ww, int &hh) const`  
  Sets width `ww` and height `hh` accordingly with the label size.

- bool `needs_keyboard () const`  
  Returns whether this widget needs a keyboard.

- void `needs_keyboard (bool needs)`  
  Sets whether this widget needs a keyboard.

- unsigned int `output () const`  
  Returns if a widget is used for output only.

- Fl_Group * `parent () const`  
  Returns a pointer to the parent widget.

- void `parent (Fl_Group *p)`  
  Internal use only - "for hacks only".

- void `position (int X, int Y)`  
  Repositions the window or widget.

- void `redraw ()`  
  Schedules the drawing of the widget.

- void `redraw_label ()`  
  Schedules the drawing of the label.

- Fl_Color `selection_color () const`  
  Gets the selection color.

- void `selection_color (Fl_Color a)`  
  Sets the selection color.

- void `set_active ()`  
  Marks the widget as active without sending events or changing focus.

- void `set_changed ()`  
  Marks the value of the widget as changed.

- void `set_output ()`  
  Sets a widget to output only.

- void `set_visible ()`  
  Makes the widget visible.

- void `set_visible_focus ()`  
  Enables keyboard focus navigation with this widget.

- int `shortcut_label () const`  
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.

- void `shortcut_label (int value)`  
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.

- virtual void `show ()`  
  Makes a widget visible.

- void `size (int W, int H)`  
  Changes the size of the widget.

- void `take_focus ()`  
  Gives the widget the keyboard focus.

- unsigned int `takesevents () const`  
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.
• const char * tooltip () const
  Gets the current tooltip text.
• void tooltip (const char *text)
  Sets the current tooltip text.
• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.
Protected Member Functions

- **Cell * add_cell** (int row, int col)
- virtual void draw () FL_OVERRIDE
  
  Draws the Fl_Grid widget and all children.
- virtual void draw_grid ()
  
  Draws the grid helper lines for design and debugging purposes.
- void init ()
- void on_remove (int) FL_OVERRIDE
  
  Fl_Group calls this method when a child widget is about to be removed.
- void remove_cell (int row, int col)

Protected Member Functions inherited from Fl_Group

- **Fl_Rect * bounds ()**
  
  Returns the internal array of widget sizes and positions.
- void draw () FL_OVERRIDE
  
  Draws the widget.
- void draw_child (Fl_Widget &widget) const
  
  Forces a child to redraw.
- void draw_children ()
  
  Draws all children of the group.
- void draw_outside_label (const Fl_Widget &widget) const
  
  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)
  
  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)
  
  Allow derived groups to act when a widget is moved within the group.
- int * sizes ()
  
  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const
  
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  
  Clears a flag in the flags mask.
- void draw_backdrop () const
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.
- void draw_label () const
Draws the widget’s label at the defined label position.

- **void draw_label**(int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.

- **Fl_Widget**(int x, int y, int w, int h, const char *label=0L)
  
  Creates a widget at the given position and size.

- unsigned int **flags**( ) const
  
  Gets the widget flags mask.

- **void h**(int v)
  
  Internal use only.

- **void set_flag**(unsigned int c)
  
  Sets a flag in the flags mask.

- **void w**(int v)
  
  Internal use only.

- **void x**(int v)
  
  Internal use only.

- **void y**(int v)
  
  Internal use only.

Protected Attributes

- bool **draw_grid_**

- **Fl_Color grid_color**

Friends

- class **Fl_Grid_Type**

Additional Inherited Members

**Static Public Member Functions inherited from Fl_Group**

- static **Fl_Group** +current()
  
  Returns the currently active group.

- static void **current**(Fl_Group *g)
  
  Sets the current group.

**Static Public Member Functions inherited from Fl_Widget**

- static void **default_callback**(Fl_Widget *widget, void *data)
  
  The default callback for all widgets that don’t set a callback.

- static unsigned int **label_shortcut**(const char *t)
  
  Returns the Unicode value of the ‘&x’ shortcut in a given text.

- static int **test_shortcut**(const char *, const bool require_alt=false)
  
  Returns true if the given text t contains the entered ‘&x’ shortcut.

**Protected Types inherited from Fl_Widget**

- enum {
  
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,

  OVERRIDE = 1<<8, VISIBLE_FOCUSE = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,

  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15

  }
**GROUP_RELATIVE = 1<<16**, **COPYED_TOOLTIP = 1<<17**, **FULLSCREEN = 1<<18**, **MAC_USE_ACCENTS_MENU = 1<<19**, **NEEDS_KEYBOARD = 1<<20**, **IMAGE_BOUND = 1<<21**, **DEIMAGE_BOUND = 1<<22**, **AUTO_DELETE_USER_DATA = 1<<23**, **MAXIMIZED = 1<<24**, **USERFLAG3 = 1<<25**, **USERFLAG2 = 1<<26**, **USERFLAG1 = 1<<27**

flags possible values enumeration.

### 12.56.1 Detailed Description

**Fl_Grid** is a container (layout) widget with multiple columns and rows. This container widget features very flexible layouts in columns and rows w/o the need to position each child widget in x/y coordinates. Widgets are assigned to grid cells (column, row) with their minimal sizes in \( w() \) and \( h() \). The \( x() \) and \( y() \) positions are ignored and can be \((0, 0)\). **Fl_Grid** calculates widget positions and resizes the widgets to fit into the grid. It is possible to create a single row or column of widgets with **Fl_Grid**.

You should design your grid with the smallest possible sizes of all widgets in mind. **Fl_Grid** will automatically assign additional space to cells according to some rules (described later) when resizing the **Fl_Grid** widget. **Hint**: You should set a minimum window size to make sure the **Fl_Grid** is never resized below its minimal sizes. Resizing below the given widget sizes results in undefined behavior. **Fl_Grid** and other container widgets (e.g. **Fl_Group**) can be nested. One main advantage of this usage is that widget coordinates in embedded **Fl_Grid** widgets become relative to the group and will be positioned as expected.

**Todo** This (relative group coordinates of nested groups of **Fl_Grid**) needs explanation and maybe an example.

**Fl_Grid** child widgets are handled by its base class **Fl_Group** but **Fl_Grid** stores additional data corresponding to each widget in internal grid cells. **Fl_Grid** children are allowed to span multiple columns and rows like HTML `<table>` cells. Individual children can have fixed sizes or be aligned inside their cells (left, right, top, bottom, and more) and/or follow their cell sizes when the **Fl_Grid** container is resized.

Note to resizing: since **Fl_Grid** uses its own layout algorithm the normal **Fl_Group::resizable()** widget is ignored (if set). Calling `init_sizes()` is not necessary.

Note

**Fl_Grid** is, as of FLTK 1.4.0, still in experimental state and should be used with caution. The API can still be changed although it is assumed to be almost stable - as stable as possible for a first release.

**Example:** Simple 3x3 **Fl_Grid** with five buttons:

```c++
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Grid.H>
#include <FL/Fl_Button.H>

int main(int argc, char *argv) {
    Fl_Double_Window *win = new Fl_Double_Window(320, 180, "3x3 Fl_Grid with Buttons");
    // create the Fl_Grid container with five buttons
    Fl_Grid *grid = new Fl_Grid(0, 0, win->w(), win->h());
    grid->layout(3, 3, 10, 10);
    grid->color(FL_WHITE);
    Fl_Button *b0 = new Fl_Button(0, 0, 0, 0, "New");
    Fl_Button *b1 = new Fl_Button(0, 0, 0, 0, "Options");
    Fl_Button *b3 = new Fl_Button(0, 0, 0, 0, "About");
    Fl_Button *b4 = new Fl_Button(0, 0, 0, 0, "Help");
    Fl_Button *b6 = new Fl_Button(0, 0, 0, 0, "Quit");
    // assign buttons to grid positions
    grid->widget(b0, 0, 0, 0, 0, "New");
    grid->widget(b3, 1, 1, 0, 0, "Options");
    grid->widget(b4, 2, 0, 0, 0, "About");
    grid->widget(b6, 2, 2, 0, 0, "Quit");
    // 1 to display grid helper lines
    grid->show_grid(1);
    win->end();
    win->resizable(grid);
    win->size_range(300, 100);
    win->show(argc, argv);
    return Fl::run();
}
```

Generated by Doxygen
12.56.2 Constructor & Destructor Documentation

12.56.2.1 Fl_Grid()

```
Fl_Grid::Fl_Grid (
    int X,
    int Y,
    int W,
    int H,
    const char ∗ L = 0 )
```

Create a new Fl_Grid widget.

**Todo**  More documentation of Fl_Grid constructor?

12.56.3 Member Function Documentation

12.56.3.1 cell() [1/2]

```
Fl_Grid::Cell ∗ Fl_Grid::cell (  
    Fl_Widget ∗ widget ) const
```

Get the grid cell of widget widget.

The pointer to the cell can be used for further assignment of properties like alignment etc.

Hint: If you know the row and column index of the cell you should use Fl_Grid::cell(int row, int col) instead because it is much faster.

Please see Fl_Grid::cell(int row, int col) for details and the validity of cell pointers.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>widget</th>
<th>widget whose cell is requested</th>
</tr>
</thead>
</table>
Return values

\textbf{NULL} if widget is not assigned to a cell

12.56.3.2 cell() [2/2]

\texttt{Fl\_Grid::Cell* Fl\_Grid::cell (int row, int col) const}

Get the grid cell of row \texttt{row} and column \texttt{col}.
Widgets and other attributes are organized in cells (\texttt{Fl\_Grid::Cell}).
This cell is an opaque structure (class) with some public methods. \textbf{Don't} assume anything about grid cell sizes and
ordering in memory. These are implementation details that can be changed without notice.
The validity of an \texttt{Fl\_Grid::Cell} pointer is limited. It will definitely be invalidated when the overall grid layout is
changed, for instance by calling layout(int, int).
Adding new cells beyond the current layout limits will also invalidate cell pointers but this is not (yet) implemented.
Attempts to assign widgets to out-of-bounds cells are currently ignored.
The only well-defined usage of cell pointers is to set one or more properties like widget alignment of a cell after
retrieving the cell pointer. Don't store cell pointers in your program for later reference.

Parameters

\begin{tabular}{ll}
  \textbf{in} & \texttt{row} & row index \\
  \textbf{in} & \texttt{col} & column index \\
\end{tabular}

Returns

pointer to cell

Return values

\textbf{NULL} if \texttt{row} or \texttt{col} is out of bounds or no widget was assigned

12.56.3.3 clear_layout()

\texttt{void Fl\_Grid::clear\_layout ( ) [virtual]}

Reset the layout w/o removing widgets.
Removes all cells and sets rows and cols to zero. Existing widgets are kept as children of the \texttt{Fl\_Group} (base class)
but are hidden.
This method should be rarely used. You may want to call \texttt{Fl\_Grid::clear()} to remove all widgets and reset the layout
to zero rows and columns.
You must call layout(int rows, int cols, ...) to set a new layout, allocate new cells, and assign widgets to new cells.
\textbf{Todo} \texttt{Fl\_Grid::clear()} needs to be implemented as documented above!

12.56.3.4 col_gap() [1/2]

\texttt{void Fl\_Grid::col\_gap (}
\texttt{const int* value,}
\texttt{size_t size )}

Set more than one column gaps at once.

See also

\texttt{Fl\_Grid::col\_weight(const int* value, size_t size)} for handling of the value array and size.
12.56.3.5  col_gap() [2/2]
void Fl_Grid::col_gap (  
    int col,  
    int value  )

Set the gap of column col.
Note that the gap is right of each column except the last one which is ignored. Use margin() for the right most column.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>col</th>
<th>column</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>gap size after the column</td>
</tr>
</tbody>
</table>

12.56.3.6  col_weight() [1/2]

void Fl_Grid::col_weight (  
    const int * value,  
    size_t size  )

Set the weight of more than one column.  
The values are taken from the array value and assigned sequentially to columns, starting from column 0. If the array size is too large extraneous values are ignored.  
Negative values in the array are not assigned to their columns, i.e. the existing value for the corresponding column is not changed.  

Example:
int val[] = { 0, 0, 50, -1, 1, 50, 0 };
grid->col_weight(val, sizeof(val)/sizeof(val[0]));

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>value</th>
<th>an array of column weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>the size of the array (number of values)</td>
</tr>
</tbody>
</table>

12.56.3.7  col_weight() [2/2]

void Fl_Grid::col_weight (  
    int col,  
    int value  )

Set the weight of a column.
Column and row weights are used to distribute additional space when the grid is resized beyond its defined (minimal) size. All weight values are relative and can be chosen freely. Suggested weights are in the range {0 .. 100}, 0 (zero) disables resizing of the column.  

How does it work?  
Whenever additional space (say: SPACE in pixels) is to be distributed to a set of columns the weights of all columns are added to a value SUM, then every single column width is increased by the value (in pseudo code):

\[
col\text{.width} += \text{SPACE} \times \text{col\text{.weight} / SUM}
\]

Resulting pixel values are rounded to the next integer and rounding differences are added to or subtracted from the column with the highest weight. If more columns have the same weight one of them is chosen.  

Note
If none of the columns considered for resizing have weights > 0 then Fl_Grid assigns the remaining space to an arbitrary column or to all considered columns evenly. This is implementation defined and can be changed without notice. You can avoid this situation by designing your grid with sensible sizes and weights.  

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>col</th>
<th>column number (counting from 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>weight, must be &gt;= 0</td>
</tr>
</tbody>
</table>
12.56.3.8 col_width() [1/2]

void Fl_Grid::col_width (  
    const int * value,  
    size_t size  
)  

Set minimal widths of more than one column.  
The values are taken from the array value and assigned sequentially to columns, starting from column 0. If the 
array size is too large extraneous values are ignored. Negative values in the array are not assigned to their columns, i.e. the existing value for the corresponding column is not changed.

Example:
```
int widths[] = { 0, 0, 50, -1, -1, 50, 0 };
grid->col_width(widths, sizeof(width)/sizeof(width[0]));
```

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>an array of column widths</td>
</tr>
<tr>
<td>size</td>
<td>the size of the array (number of values)</td>
</tr>
</tbody>
</table>

12.56.3.9 col_width() [2/2]

void Fl_Grid::col_width (  
    int col,  
    int value  
)  

Set the minimal width of a column.  
Column widths are calculated by using the maximum of all widget widths in that column and the given column width. After calculating the width additional space is added when resizing according to the weight of the column. You can set one or more column widths in one call by using `Fl_Grid::col_width(const int *value, size_t size)`.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>column number (counting from 0)</td>
</tr>
<tr>
<td>value</td>
<td>minimal column width, must be ≥ 0</td>
</tr>
</tbody>
</table>

See also

`Fl_Grid::col_width(const int *value, size_t size)`

12.56.3.10 debug()

void Fl_Grid::debug (  
    int level = 127  
)  

Output layout information of this Fl_Grid to stderr. Parameter level will be used to define the amount of output.

- 0 = nothing  
- 127 = everything  
- other values not yet defined

Note

It is not yet defined which kind of values level will have, either a numerical value (127 = maximum, 0 = nothing) or a bit mask that determines what to output.

Todo Add more information about cells and children.

Control output by using level.
Parameters

| in  | level      | not yet used (0-127, default = 127) |

### 12.56.3.11 draw()

```cpp
void Fl_Grid::draw () [protected], [virtual]
```

Draws the Fl_Grid widget and all children.
If the layout has been changed **layout()** is called before the widget is drawn so all children are arranged as designed.

See also

- **layout()**
- **need_layout()**

Implements **Fl_Widget**.

### 12.56.3.12 draw_grid()

```cpp
void Fl_Grid::draw_grid () [protected], [virtual]
```

Draws the grid helper lines for design and debugging purposes. This method is protected so it can be modified in subclasses.

### 12.56.3.13 gap() [1/2]

```cpp
void Fl_Grid::gap ( int * row_gap, int * col_gap ) const
```

Get the default gaps for rows and columns.

Parameters

| out  | row_gap | pointer to int to receive column gap, may be NULL |
| out  | col_gap | pointer to int to receive column gap, may be NULL |

### 12.56.3.14 gap() [2/2]

```cpp
void Fl_Grid::gap ( int row_gap, int col_gap = -1 ) [virtual]
```

Set default gaps for rows and columns.

All gaps are positioned below the rows and right of their columns. The bottom row and the right-most column don't have a gap, i.e. the gap sizes of these columns and rows are ignored. You can use a right or bottom margin instead.

You have to specify at least one argument, **col_gap** is optional. If you don't specify an argument or use a negative value (e.g. -1) then that margin is not affected.

You can also initialize the default gaps with **layout(int, int, int)**.

Parameters

| in  | row_gap | default gap for all rows |
| in  | col_gap | default gap for all columns |
See also

`Fl_Grid::layout(int rows, int cols, int margin, int gap)`

12.56.3.15  `layout()` [1/2]

```cpp
void Fl_Grid::layout ( ) [virtual]
```

Calculate the grid layout and resize and position all widgets.
This is called automatically when the `Fl_Grid` is resized. You need to call it once after you added widgets or moved widgets between cells.
Calling it once after all modifications are completed is enough.

**Todo**  Document when and why to call `layout()` w/o args. See `Fl_Flex::layout()`

See also

`Fl_Grid::layout(int rows, int cols, int margin, int gap)`

12.56.3.16  `layout()` [2/2]

```cpp
void Fl_Grid::layout ( int rows,
     int cols, int margin = -1,
     int gap = -1 ) [virtual]
```

Set the basic layout parameters of the `Fl_Grid` widget.
You need to specify at least `rows` and `cols` to define a layout before you can add widgets to the grid.
Parameters `margin` and `gap` are optional.
You can call `layout(int rows, int cols, int margin, int gap)` again to change the layout but this is inefficient since all cells are reallocated if the layout changed.
Calling this with the same values of `rows` and `cols` is fast and can be used to change `margin` and `gap` w/o reallocating the cells.

`margin` sets all margins (left, top, right, bottom) to the same value. Negative values (e.g. -1) don’t change the established margins. The default value set by the constructor is 0.
`gap` sets row and column gaps to the same value. Negative values (e.g. -1) do not affect the established gaps. The default value set by the constructor is 0.
After you added all widgets you must call `layout()` once without arguments to calculate the actual layout and to position and resize all widgets.

**Todo**  Document when and why to call `layout()` w/o args. See `Fl_Flex::layout()`

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><code>rows</code></td>
<td>number of rows</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>cols</code></td>
<td>number of columns</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>margin</code></td>
<td>margin size inside the <code>Fl_Grid</code>'s border</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>gap</code></td>
<td>gap size between cells</td>
</tr>
</tbody>
</table>

See also

`Fl_Grid::layout()`

12.56.3.17  `margin()` [1/2]

```cpp
int Fl_Grid::margin ( int * left,  
                        int * top,  
                        int * right,  
                        int * bottom,  
                        int margin ) [virtual]
```

Set the margin size of the `Fl_Grid` widget.
You need to specify at least `left`, `top`, `right`, and `bottom` to define a layout before you can add widgets to the grid.
Parameters `margin` and `gap` are optional.
You can call `margin(int * left, int * top, int * right, int * bottom, int margin)` again to change the layout but this is inefficient since all cells are reallocated if the layout changed.
Calling this with the same values of `left`, `top`, `right`, and `bottom` is fast and can be used to change `margin` w/o reallocating the cells.

After you added all widgets you must call `margin()` once without arguments to calculate the actual layout and to position and resize all widgets.

**Todo**  Document when and why to call `margin()` w/o args. See `Fl_Flex::margin()`

See also

`Fl_Grid::margin()`
Returns all outside margin sizes of the grid.
All margin sizes are returned in the given arguments. If any argument is NULL the respective value is not returned.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>left</th>
<th>returns left margin if not NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>top</td>
<td>returns top margin if not NULL</td>
</tr>
<tr>
<td>out</td>
<td>right</td>
<td>returns right margin if not NULL</td>
</tr>
<tr>
<td>out</td>
<td>bottom</td>
<td>returns bottom margin if not NULL</td>
</tr>
</tbody>
</table>

Returns

whether all margins are equal

Return values

| 1 | all margins have the same size |
| 0 | at least one margin has a different size |

### 12.56.3.18 margin() [2/2]

void Fl_Grid::margin (  
  int left,  
  int top = -1,  
  int right = -1,  
  int bottom = -1 ) [virtual]  

Set all margins (left, top, right, bottom).
All margins are measured in pixels inside the box borders. You need to specify at least one argument, all other arguments are optional. If you don't specify an argument or use a negative value (e.g. -1) then that particular margin is not affected.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>left</th>
<th>left margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>top</td>
<td>top margin</td>
</tr>
<tr>
<td>in</td>
<td>right</td>
<td>right margin</td>
</tr>
<tr>
<td>in</td>
<td>bottom</td>
<td>bottom margin</td>
</tr>
</tbody>
</table>

### 12.56.3.19 need_layout()

void Fl_Grid::need_layout (  
  int set ) [inline]  

Request or reset the request to calculate the layout of children.  
If called with true (1) this calls redraw() to schedule a full draw(). When draw is eventually called, the layout is (re)calculated before actually drawing the widget.

Parameters

| in | set | 1 to request layout calculation, 0 to reset the request |
12.56.3.20 on_remove()

```c
void Fl_Grid::on_remove (  
    int index  ) [protected], [virtual]
```

*Fl_Group* calls this method when a child widget is about to be removed. Make sure that the widget is also removed from our internal list of children. Reimplemented from *Fl_Group*.

12.56.3.21 resize()

```c
void Fl_Grid::resize (  
    int X,  
    int Y,  
    int W,  
    int H ) [virtual]
```

Recalculate the layout and position and resize all widgets. This method overrides *Fl_Group::resize()* and calculates all positions and sizes of its children according to its own rules.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X, Y</th>
<th>new widget position</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>W, H</td>
<td>new widget size</td>
</tr>
</tbody>
</table>

Reimplemented from *Fl_Widget*.

12.56.3.22 row_gap() [1/2]

```c
void Fl_Grid::row_gap (  
    const int * value,  
    size_t size       )
```

Set more than one row gaps at once.

See also

*Fl_Grid::col_weight(const int *value, size_t size)* for handling of the value array and *size*.

12.56.3.23 row_gap() [2/2]

```c
void Fl_Grid::row_gap (  
    int row,  
    int value )
```

Set the gap of row *row*.

Note that the gap is below each row except the last one which is ignored. Use *margin()* for the bottom row.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>row</th>
<th>row</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>gap size below the row</td>
</tr>
</tbody>
</table>

12.56.3.24 row_height() [1/2]

```c
void Fl_Grid::row_height (  
    const int * value,  
    size_t size       )
```

Set the minimal row height of more than one row.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>value</th>
<th>array of height values</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>size of array value</td>
</tr>
</tbody>
</table>

See also

`Fl_Grid::col_weight(const int *value, size_t size)` for handling of the value array and size.

12.56.3.25  `row_height()` [2/2]

```cpp
void Fl_Grid::row_height ( int row, int value )
```

Set the minimal row height of row `row`.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>row</th>
<th>row</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>minimal height of the row</td>
</tr>
</tbody>
</table>

12.56.3.26  `row_weight()` [1/2]

```cpp
void Fl_Grid::row_weight ( const int *value, size_t size )
```

Set the weight of more than one row.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>value</th>
<th>array of height values</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>size of array value</td>
</tr>
</tbody>
</table>

See also

`Fl_Grid::col_weight(const int *value, size_t size)` for handling of the value array and size.

12.56.3.27  `row_weight()` [2/2]

```cpp
void Fl_Grid::row_weight ( int row, int value )
```

Set the row weight of row `row`.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>row</th>
<th>row</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>weight of the row</td>
</tr>
</tbody>
</table>

12.56.3.28  `show_grid()` [1/2]

```cpp
void Fl_Grid::show_grid ( int set ) [inline]
```
Enable or disable drawing of the grid helper lines for visualization. Use this method during the design stage of your Fl_Grid widget or for debugging if widgets are not positioned as intended. The default is a light green color but you can change it for better contrast if needed, see show_grid(int set, Fl_Color col).

Note

You can define the environment variable FLTK_GRID_DEBUG=1 to set show_grid(1) for all Fl_Grid widgets at construction time. This enables you to debug the grid layout w/o changing code.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set</code></td>
<td>1 (true) = draw, 0 = don't draw the grid</td>
</tr>
</tbody>
</table>

See also

show_grid(int set, Fl_Color col)

12.56.3.29 show_grid() [2/2]

void Fl_Grid::show_grid ( 
    int set, 
    Fl_Color col ) [inline]

Enable or disable drawing of the grid helper lines for visualization. This method also sets the color used for the helper lines. The default is a light green color but you can change it to any color for better contrast if needed.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set</code></td>
<td>1 (true) = draw, 0 = don't draw the grid</td>
</tr>
<tr>
<td><code>col</code></td>
<td>color to use for the grid helper lines</td>
</tr>
</tbody>
</table>

See also

show_grid(int set)

12.56.3.30 widget() [1/2]

Fl_Grid::Cell * Fl_Grid::widget ( 
    Fl_Widget * wi, 
    int row, 
    int col, 
    Fl_Grid_Align align = FL_GRID_FILL )

Assign a widget to a grid cell and set its alignment. This short form sets row and column spanning attributes to (1, 1). For more information see Fl_Grid::widget(Fl_Widget *wi, int row, int col, int rowspan, int colspan, Fl_Grid_Align align)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wi</code></td>
<td>widget to be assigned to the cell</td>
</tr>
<tr>
<td><code>row</code></td>
<td>row</td>
</tr>
<tr>
<td><code>col</code></td>
<td>column</td>
</tr>
<tr>
<td><code>align</code></td>
<td>widget alignment inside the cell</td>
</tr>
</tbody>
</table>
Assign a widget to a grid cell and set cell spanning and alignment. Default alignment is FL_GRID_FILL which stretches the widget in horizontal and vertical directions to fill the whole cell(s) given by colspan and rowspan. You can use this method to move a widget from one cell to another; it is automatically removed from its old cell. If the new cell is already assigned to another widget that widget is deassigned but kept as a child of the group. Before you can assign a widget to a cell it must have been created as a child of the Fl_Grid widget (i.e. its Fl_Group).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wi</td>
<td>widget to be assigned to the cell</td>
</tr>
<tr>
<td>row</td>
<td>row</td>
</tr>
<tr>
<td>col</td>
<td>column</td>
</tr>
<tr>
<td>rowspan</td>
<td>vertical span in cells, default 1</td>
</tr>
<tr>
<td>colspan</td>
<td>horizontal span in cells, default 1</td>
</tr>
<tr>
<td>align</td>
<td>widget alignment inside the cell</td>
</tr>
</tbody>
</table>

Returns

assigned cell

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>if row or col is out of bounds</td>
</tr>
</tbody>
</table>

See also

Fl_Grid::widget(Fl_Widget *wi, int row, int col, int rowspan, int colspan, Fl_Grid_Align align)

The documentation for this class was generated from the following files:

- Fl_Grid.H
- Fl_Grid.cxx

12.57 Fl_Group Class Reference

The Fl_Group class is the FLTK container widget.

#include <Fl_Group.H>

Inheritance diagram for Fl_Group:
Public Member Functions

- **FL_Widget** *&* _ddfdesign_kludge ()
  
  *This is for forms compatibility only.*

- **void add (FL_Widget &)*

  *The widget is removed from its current group (if any) and then added to the end of this group.*

- **void add (FL_Widget *o)*

  *See void FL_Group::add(FL_Widget &w)*

- **void add_resizable (FL_Widget *o)*

  *Adds a widget to the group and makes it the resizable widget.*

- **FL_Widget *const + array () const*

  *Returns a pointer to the array of children.*

- **FL_Group const + as_group () const FL_OVERRIDE*

  *Returns an FL_Group pointer if this widget is an FL_Group.*

- **void begin ()*

  *Sets the current group so you can build the widget tree by just constructing the widgets.*

- **FL_Widget *child (int n) const***

  *Returns array()[n].*

- **int children () const*

  *Returns how many child widgets the group has.*

- **void clear ()*

  *Deletes all child widgets from memory recursively.*

- **unsigned int clip_children ()***

  *Returns the current clipping mode.*
• void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.
• virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.
• void end ()
  Exactly the same as current(this->parent()).
• int find (const Fl_Widget &o) const
  See int Fl_Group::find(const Fl_Widget *w) const.
• int find (const Fl_Widget *) const
  Searches the child array for the widget and returns the index.
• Fl_Group (int, int, int, int, const char * = 0)
  Creates a new Fl_Group widget using the given position, size, and label string.
• void focus (Fl_Widget *W)
• void forms_end ()
  This is for forms compatibility only.
• int handle (int) FL_OVERRIDE
  Handles the specified event.
• void init_sizes ()
  Resets the internal array of widget sizes and positions.
• void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.
• void insert (Fl_Widget &, Fl_Widget *before)
  This does insert(w, find(before)).
• void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.
• void remove (Fl_Widget *o)
  Removes the widget o from the group.
• void remove (int index)
  Removes the widget at index from the group but does not delete it.
• Fl_Widget *resizable () const
  Returns the group's resizable widget.
• void resizable (Fl_Widget &o)
  Sets the group's resizable widget.
• void resizable (Fl_Widget *o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
• void resize (int, int, int, int) FL_OVERRIDE
  Resizes the Fl_Group widget and all of its children.
• virtual ~Fl_Group ()
  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget
• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  Activates the widget.
• unsigned int active () const
  Returns whether the widget is active.
• int active_r () const
  Returns whether the widget and all of its parents are active.
Gets the label alignment.

- `void align(Fl_Align alignment)`
  Sets the label alignment.

- `long argument() const`  
  Gets the current user data (long) argument that is passed to the callback function.

- `void argument(long v)`  
  Sets the current user data (long) argument that is passed to the callback function.

- `virtual class Fl_Gl_Window * as_gl_window()`  
  Returns an `Fl_Gl_Window` pointer if this widget is an `Fl_Gl_Window`.

- `virtual class Fl_Gl_Window const * as_gl_window() const`  
  Returns an `Fl_Gl_Window` pointer if this widget is an `Fl_Gl_Window`.

- `virtual Fl_Window * as_window()`  
  Returns an `Fl_Window` pointer if this widget is an `Fl_Window`.

- `virtual Fl_Window const * as_window() const`  
  Returns an `Fl_Window` pointer if this widget is an `Fl_Window`.

- `void bind_deimage(Fl_Image * img)`  
  Sets the image to use as part of the widget label when in the inactive state.

- `void bind_deimage(int f)`  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- `void bind_image(Fl_Image * img)`  
  Sets the image to use as part of the widget label when in the active state.

- `void bind_image(int f)`  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- `Fl_Boxtype box() const`  
  Gets the box type of the widget.

- `void box(Fl_Boxtype new_box)`  
  Sets the box type for the widget.

- `Fl_Callback_p callback() const`  
  Gets the current callback function for the widget.

- `void callback(Fl_Callback * cb)`  
  Sets the current callback function for the widget.

- `void callback(Fl_Callback * cb, Fl_Callback_User_Data * p, bool auto_free)`  
  Sets the current callback function and managed user data for the widget.

- `void callback(Fl_Callback * cb, void * p)`  
  Sets the current callback function and data for the widget.

- `void callback(Fl_Callback0 * cb)`  
  Sets the current callback function for the widget.

- `void callback(Fl_Callback1 * cb, long p=0)`  
  Sets the current callback function for the widget.

- `unsigned int changed() const`  
  Checks if the widget value changed since the last callback.

- `void clear_active()`  
  Marks the widget as inactive without sending events or changing focus.

- `void clear_changed()`  
  Marks the value of the widget as unchanged.

- `void clear_damage(uchar c=0)`  
  Clears or sets the damage flags.

- `void clear_output()`  
  Sets a widget to accept input.

- `void clear_visible()`  
  Hides the widget.

- `void clear_visible_focus()`  
  Disables keyboard focus navigation with this widget.
- **Fl_Color color ( ) const**
  
  Gets the background color of the widget.

- **void color (Fl_Color bg)**
  
  Sets the background color of the widget.

- **void color (Fl_Color bg, Fl_Color sel)**
  
  Sets the background and selection color of the widget.

- **Fl_Color color2 ( ) const**
  
  For back compatibility only.

- **void color2 (unsigned a)**
  
  For back compatibility only.

- **int contains (const Fl_Widget *w) const**
  
  Checks if w is a child of this widget.

- **void copy_label (const char *new_label)**
  
  Sets the current label.

- **void copy_tooltip (const char *text)**
  
  Sets the current tooltip text.

- **uchar damage ( ) const**
  
  Returns non-zero if draw() needs to be called.

- **void damage (uchar c)**
  
  Sets the damage bits for the widget.

- **void damage (uchar c, int x, int y, int w, int h)**
  
  Sets the damage bits for an area inside the widget.

- **int damage_resize (int, int, int, int)**
  
  Internal use only.

- **void deactivate ( )**
  
  Deactivates the widget.

- **Fl_Image * deimage ( )**
  
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image * deimage ( ) const**
  
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (Fl_Image &img)**
  
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (Fl_Image *img)**
  
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound ( ) const**
  
  Returns whether the inactive image is managed by the widget.

- **void do_callback (Fl_Callback_Reas one FL_REASON_UNKNOWN)**

  Calls the widget callback function with default arguments.

- **void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reas one FL_REASON_UNKNOWN)**

  Calls the widget callback function with arbitrary arguments.

- **void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reas one FL_REASON_UNKNOWN)**

  Calls the widget callback function with arbitrary arguments.

- **void draw_label (int, int, int, int, Fl_Align) const**

  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h ( ) const**

  Gets the widget height.

- **virtual void hide ( )**

  Makes a widget invisible.

- **Fl_Image * image ( )**

  Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image * image ( ) const**
Gets the image that is used as part of the widget label when in the active state.

• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.

• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.

• int image_bound () const
  Returns whether the image is managed by the widget.

• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.

• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().

• const char * label () const
  Gets the current label text.

• void label (const char *text)
  Sets the current label pointer.

• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.

• Fl_Color labelcolor () const
  Gets the label color.

• void labelcolor (Fl_Color c)
  Sets the label color.

• Fl_Font labelfont () const
  Gets the font to use.

• void labelfont (Fl_Font f)
  Sets the font to use.

• Fl_Fontsize labelsize () const
  Gets the font size in pixels.

• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.

• Fl_Labeltype labeltype () const
  Gets the label type.

• void labeltype (Fl_Labeltype a)
  Sets the label type.

• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.

• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.

• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.

• unsigned int output () const
  Returns if a widget is used for output only.

• Fl_Group * parent () const
  Returns a pointer to the parent widget.

• void parent (Fl_Group *p)
  Internal use only - "for hacks only".

• void position (int X, int Y)
  Repositions the window or widget.

• void redraw ()
  Schedules the drawing of the widget.

• void redraw_label ()
  Schedules the drawing of the label.
- `Fl_Color selection_color () const`
  
  Gets the selection color.

- `void selection_color (Fl_Color a)`
  
  Sets the selection color.

- `void set_active ()`
  
  Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  
  Marks the value of the widget as changed.

- `void set_output ()`
  
  Sets a widget to output only.

- `void set_visible ()`
  
  Makes the widget visible.

- `void set_visible_focus ()`
  
  Enables keyboard focus navigation with this widget.

- `int shortcut_label () const`
  
  Returns whether the widget's label uses '&' to indicate shortcuts.

- `void shortcut_label (int value)`
  
  Sets whether the widget's label uses '&' to indicate shortcuts.

- `virtual void show ()`
  
  Makes a widget visible.

- `void size (int W, int H)`
  
  Changes the size of the widget.

- `int take_focus ()`
  
  Gives the widget the keyboard focus.

- `unsigned int takesevents () const`
  
  Returns if the widget is able to take events.

- `int testShortcut ()`
  
  Returns true if the widget's label contains the entered 'x' shortcut.

- `const char * tooltip () const`
  
  Gets the current tooltip text.

- `void tooltip (const char * text)`
  
  Sets the current tooltip text.

- `Fl_Window * top_window () const`
  
  Returns a pointer to the top-level window for the widget.

- `Fl_Window * top_window_offset (int &xoff, int &yoff) const`
  
  Finds the x/y offset of the current widget relative to the top-level window.

- `uchar type () const`
  
  Gets the widget type.

- `void type (uchar t)`
  
  Sets the widget type.

- `int use_accents_menu ()`
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- `void * user_data () const`
  
  Gets the user data for this widget.

- `void user_data (Fl_Callback_User_Data *v, bool auto_free)`
  
  Sets the user data for this widget.

- `void user_data (void *v)`
  
  Sets the user data for this widget.

- `unsigned int visible () const`
  
  Returns whether a widget is visible.

- `unsigned int visible_focus () const`
Checks whether this widget has a visible focus.

- void **visible_focus**( int v)
  
  Modifies keyboard focus navigation.

- int **visible_r** () const

  Returns whether a widget and all its parents are visible.

- int **w** () const

  Gets the widget width.

- **Fl_When** when () const

  Returns the conditions under which the callback is called.

- void **when** (uchar i)

  Sets the flags used to decide when a callback is called.

- **Fl_Window** **∗** window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.

- int **x** () const

  Gets the widget position in its window.

- int **y** () const

  Gets the widget position in its window.

- virtual **∼Fl_Widget** ()

  Destroys the widget.

**Static Public Member Functions**

- static **Fl_Group** **∗** current ()

  Returns the currently active group.

- static void current (Fl_Group ∗g)

  Sets the current group.

**Static Public Member Functions inherited from Fl_Widget**

- static void default_callback (Fl_Widget ∗widget, void ∗data)

  The default callback for all widgets that don't set a callback.

- static unsigned int labelShortcut (const char ∗t)

  Returns the Unicode value of the '&x' shortcut in a given text.

- static int testShortcut (const char ∗t, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

**Protected Member Functions**

- Fl_Rect **∗** bounds ()

  Returns the internal array of widget sizes and positions.

- void **draw** () FL_OVERRIDE

  Draws the widget.

- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.

- void draw_children ()

  Draws all children of the group.

- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.

- virtual int on_insert (Fl_Widget ∗, int)

  Allow derived groups to act when a widget is added as a child.

- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
• virtual void on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.
• int * sizes ()
  Returns the internal array of widget sizes and positions.
• void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const
  Draws the widget box according its box style.
• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
• void draw_focus () const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
• void draw_label () const
  Draws the widget's label at the defined label position.
• void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
• Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.
• unsigned int flags () const
  Gets the widget flags mask.
• void h (int v)
  Internal use only.
• void set_flag (unsigned int c)
  Sets a flag in the flags mask.
• void w (int v)
  Internal use only.
• void x (int v)
  Internal use only.
• void y (int v)
  Internal use only.
Additional Inherited Members

Protected Types inherited from **Fl_Widget**

• enum {
  
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTip = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22, 
  AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }

flags possible values enumeration.

12.57.1 Detailed Description

The **Fl_Group** class is the FLTK container widget. It maintains an array of child widgets. These children can themselves be any widget including **Fl_Group**. The most important subclass of **Fl_Group** is **Fl_Window**, however groups can also be used to control radio buttons or to enforce resize behavior.

The tab and arrow keys are used to move the focus between widgets of this group, and to other groups. The only modifier grabbed is shift (for shift-tab), so that ctrl-tab, alt-up, and such are free for the app to use as shortcuts.

To remove a widget from the group and destroy it, in 1.3.x (and up) you can simply use:

```
delete some_widget;
```

..and this will trigger proper scheduling of the widget's removal from its parent group.

If used as a child of **Fl_Tabs**, setting `when(FL_WHEN_CLOSED)` will enable the Close button in the corresponding tab. If the user clicks the Close button, the callback of this group will be called with the callback reason `FL_REASON_CLOSED`.

12.57.2 Constructor & Destructor Documentation

12.57.2.1 **Fl_Group()**

```
Fl_Group::Fl_Group ( 
  int X,
  int Y,
  int W,
  int H,
  const char * l = 0 )
```

Creates a new **Fl_Group** widget using the given position, size, and label string. The default boxtype is **FL_NO_BOX**.

12.57.2.2 **~Fl_Group()**

```
Fl_Group::~Fl_Group ( ) [virtual]
```

The destructor also deletes all the children.

This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. It is allowed that the **Fl_Group** and all of its children are automatic (local) variables, but you must declare the **Fl_Group** first, so that it is destroyed last.

If you add static or automatic (local) variables to an **Fl_Group**, then it is your responsibility to remove (or delete) all such static or automatic child widgets **before destroying** the group - otherwise the group will attempt to call delete operator on them leading to undefined behavior!
12.57.3 Member Function Documentation

12.57.3.1 array()

```cpp
Fl_Widget *const * Fl_Group::array () const
```

Returns a pointer to the array of children.

**Note**

This pointer is only valid until the next time a child is added or removed.

12.57.3.2 as_group() [1/2]

```cpp
Fl_Group const * Fl_Group::as_group ( ) const [inline], [virtual]
```

Reimplemented from `Fl_Widget`.

12.57.3.3 as_group() [2/2]

```cpp
Fl_Group * Fl_Group::as_group ( ) [inline], [virtual]
```

Returns an `Fl_Group` pointer if this widget is an `Fl_Group`. Use this method if you have a widget (pointer) and need to know whether this widget is derived from `Fl_Group`. If it returns non-NULL, then the widget in question is derived from `Fl_Group`, and you can use the returned pointer to access its children or other `Fl_Group`-specific methods.

**Example:**

```cpp
void my_callback (Fl_Widget *w, void *) {
  Fl_Group *g = w->as_group();
  if (g)
    printf ("This group has %d children\n", g->children());
  else
    printf ("This widget is not a group!\n");
}
```

**Return values**

- `NULL` if this widget is not derived from `Fl_Group`.

**Note**

This method is provided to avoid `dynamic_cast`.

**See also**

- `Fl_Widget::as_window()`, `Fl_Widget::as_gl_window()`

Reimplemented from `Fl_Widget`.

12.57.3.4 begin()

```cpp
void Fl_Group::begin ( )
```

Sets the current group so you can build the widget tree by just constructing the widgets. `begin()` is automatically called by the constructor for `Fl_Group` (and thus for `Fl_Window` as well). `begin()` is exactly the same as `current(this)`. Don't forget to `end()` the group or window!

12.57.3.5 bounds()

```cpp
Fl_Rect * Fl_Group::bounds ( ) [protected]
```

Returns the internal array of widget sizes and positions. If the `bounds()` array does not exist, it will be allocated and filled with the current widget sizes and positions. The `bounds()` array stores the initial positions of widgets as `Fl_Rect`'s. The size of the array is `children() + 2`.

- The first `Fl_Rect` is the group,
- the second is the resizable (clipped to the group),

Generated by Doxygen
- the rest are the children.

This is a convenient order for the resize algorithm. If the group and/or the resizable() is a Fl_Window (or subclass) then the x() and y() coordinates of their respective Fl_Rect's are zero.

Note
You should never need to use this protected method directly, unless you have special needs to rearrange the children of a Fl_Group. Fl_Tile uses this to rearrange its widget positions. The returned array should be considered read-only. Do not change its contents. If you need to rearrange children in a group, do so by resizing the children and call init_sizes().

#include <FL/Fl_Rect.H> if you want to access the bounds() array in your derived class. Fl_Rect.H is intentionally not included by Fl_Group.H to avoid unnecessary dependencies.

Returns
Array of Fl_Rect's with widget positions and sizes. The returned array is only valid until init_sizes() is called or widgets are added to or removed from the group.

See also
init_sizes()

Since
FLTK 1.4.0

12.57.3.6 child()

Fl_Widget * Fl_Group::child ( int n ) const [inline]

Returns array()[n]. No range checking is done!

12.57.3.7 clear()

void Fl_Group::clear ( void )

Deletes all child widgets from memory recursively. This method differs from the remove() method in that it affects all child widgets and deletes them from memory. The resizable() widget of the Fl_Group is set to the Fl_Group itself.

12.57.3.8 clip_children() [1/2]

unsigned int Fl_Group::clip_children ( ) [inline]

Returns the current clipping mode.

Returns
true, if clipping is enabled, false otherwise.

See also
void Fl_Group::clip_children(int c)

12.57.3.9 clip_children() [2/2]

void Fl_Group::clip_children ( int c ) [inline]

Controls whether the group widget clips the drawing of child widgets to its bounding box. Set c to 1 if you want to clip the child widgets to the bounding box. The default is to not clip (0) the drawing of child widgets.
12.57.3.10 current() [1/2]

`Fl_Group * Fl_Group::current ( ) [static]`

Returns the currently active group.

The `Fl_Widget` constructor automatically does `current() -> add(widget)` if this is not null. To prevent new widgets from being added to a group, call `Fl_Group::current(0).

12.57.3.11 current() [2/2]

`void Fl_Group::current ( Fl_Group * g ) [static]`

Sets the current group.

See also

`Fl_Group::current()`

12.57.3.12 delete_child()

`int Fl_Group::delete_child ( int index ) [virtual]`

Removes the widget at `index` from the group and deletes it.

This method does nothing if `index` is out of bounds.

This method differs from the `remove()` method in that it deletes the widget from memory. Since this method is virtual it can be reimplemented in subclasses with additional requirements and consequences. See the documentation of subclasses.

Many subclasses don't need to reimplement this method.

Note

This method may refuse to remove and delete the widget if it is an essential part of the `Fl_Group`, for instance a scrollbar in an `Fl_Scroll` group. In this case the widget is neither removed nor deleted.

This method does not call `init_sizes()` or `redraw()`. This is left to user code if necessary.

Returns 0 if the widget was removed and deleted. Return values > 0 are reserved for use by FLTK core widgets. Return values < 0 are free to be used by user defined widgets.

Todo Reimplementation of `Fl_Group::delete_child(int)` in more FLTK subclasses. This is not yet complete.

Parameters

| in index | index of child to be removed |

Returns

success (0) or error code

Return values

| 0 success | 1 index out of range | 2 widget not allowed to be removed (see note) | >2 reserved for FLTK use |

Since

FLTK 1.4.0

Reimplemented in `Fl_Scroll`.

Generated by Doxygen
12.57.13  draw()

void Fl_Group::draw ( ) [protected], [virtual]
Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as
soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded
scrollbar, you can do it (because draw() is virtual) like this:
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
Implements Fl_Widget.
Reimplemented in Fl_Help_View, Fl_Pack, Fl_Scroll, Fl_Spinner, Fl_Table, Fl_Tabs, Fl_Text_Display, Fl_Tree,
Fl_Window, Fl_Wizard, and Fl_Terminal.

12.57.14  draw_child()

void Fl_Group::draw_child ( 
    Fl_Widget & widget ) const [protected]
Forces a child to redraw.
This draws a child widget, if it is not clipped. The damage bits are cleared after drawing.

12.57.15  draw_children()

void Fl_Group::draw_children ( ) [protected]
Draws all children of the group.
This is useful, if you derived a widget from Fl_Group and want to draw a special border or background. You can call
draw_children() from the derived draw() method after drawing the box, border, or background.

12.57.16  end()

void Fl_Group::end ( )
Exactly the same as current(this->parent()).
Any new widgets added to the widget tree will be added to the parent of the group.

12.57.17  find()

int Fl_Group::find ( 
    const Fl_Widget * o ) const
Searches the child array for the widget and returns the index.
Returns children() if the widget is NULL or not found.

12.57.18  focus()

void Fl_Group::focus ( 
    Fl_Widget * W ) [inline]

Deprecated  This is for backwards compatibility only. You should use W->take_focus() instead.
See also
    Fl_Widget::take_focus();

12.57.19  handle()

int Fl_Group::handle ( 
    int event ) [virtual]
Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class.

In rare cases you may want to return 1 from your `handle()` method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the `handle()` method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in | event | the kind of event received |

Return values

| 0  | if the event was not used or understood |
| 1  | if the event was used and can be deleted |

See also

`Fl_Event`

Reimplemented from `Fl_Widget`.
Reimplemented in `Fl_Table, Fl_Terminal, Fl_Text_Display, Fl_Text_Editor, Fl_Tree, Fl_Spinner, Fl_Table_Row, Fl_Tile, Fl_Help_View, Fl_Scroll, Fl_Tabs, and Fl_Window`.

### 12.57.3.20 init_sizes()

```cpp
void Fl_Group::init_sizes()
```

Resets the internal array of widget sizes and positions.

The `Fl_Group` widget keeps track of the original widget sizes and positions when resizing occurs so that if you resize a window back to its original size the widgets will be in the correct places. If you rearrange the widgets in your group, call this method to register the new arrangement with the `Fl_Group` that contains them.

If you add or remove widgets, this will be done automatically.

Note

The internal array of widget sizes and positions will be allocated and filled when the next `resize()` occurs. For more information on the contents and structure of the `bounds()` array see `bounds()`.

See also

`bounds()`

`sizes()` (deprecated)

### 12.57.3.21 insert() [1/2]

```cpp
void Fl_Group::insert(
    Fl_Widget & o,
    int index)
```

The widget is removed from its current group (if any) and then inserted into this group.

It is put at index \( n \) - or at the end, if \( n \geq \text{children().} \). This can also be used to rearrange the widgets inside a group.
12.57.3.22  **insert()**  [2/2]

```cpp
void Fl_Group::insert ( 
    Fl_Widget & o,  
    Fl_Widget * before ) [inline]
```

This does insert(w, find(before)).
This will append the widget if before is not in the group.

12.57.3.23  **on_insert()**

```cpp
int Fl_Group::on_insert ( 
    Fl_Widget * candidate,  
    int index ) [protected], [virtual]
```

Allow derived groups to act when a widget is added as a child.
Widgets derived from Fl_Group may store additional data for their children. Overriding this method will allow derived classes to generate these data structures just before the child is added.
This method usually returns the same index that was given in the parameters. By setting a new index, the position of other widgets in the child pointer array can be preserved (e.g. Fl_Scroll keeps its scroll bars as the last two children).
By returning -1, Fl_Group::insert will not add the child to array_. This is not recommended, but Fl_Table does something similar to forward children to a hidden group.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>candidate</td>
</tr>
<tr>
<td>index</td>
</tr>
</tbody>
</table>

Returns

- index to position the child as planned  
- a new index to force the child to a different position  
-1 to keep the group from adding the candidate

Reimplemented in Fl_Scroll, Fl_Tabs, and Fl_Tile.

12.57.3.24  **on_move()**

```cpp
int Fl_Group::on_move ( 
    int oldIndex,  
    int newIndex ) [protected], [virtual]
```

Allow derived groups to act when a widget is moved within the group.
Widgets derived from Fl_Group may store additional data for their children. Overriding this method will allow derived classes to move these data structures just before the child itself is moved.
This method usually returns the new index that was given in the parameters. By setting a different destination index, the position of other widgets in the child pointer array can be preserved.
By returning -1, Fl_Group::insert will not move the child.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldIndex</td>
</tr>
<tr>
<td>newIndex</td>
</tr>
</tbody>
</table>

Returns

- newIndex to position the child as planned  
- a different index to force the child to a different position  
-1 to keep the group from moving the child
Reimplemented in Fl_Scroll, Fl_Tabs, and Fl_Tile.

**12.57.3.25 on_remove()**

```cpp
template<typename T>
void Fl_Group::on_remove (int index) const [protected], [virtual]
```

Allow derived groups to act when a child widget is removed from the group. Widgets derived from Fl_Group may store additional data for their children. Overriding this method will allow derived classes to remove these data structures just before the child is removed.

**Parameters**

- `index`: remove the child at this position in the array.

Reimplemented in Fl_Flex, Fl_Grid, Fl_Tabs, and Fl_Tile.

**12.57.3.26 remove() [1/3]**

```cpp
void Fl_Group::remove (Fl_Widget & o)
```

Removes a widget from the group but does not delete it. This method does nothing if the widget is not a child of the group. This method differs from the clear() method in that it only affects a single widget and does not delete it from memory.

**Note**

If you have the child's index anyway, use remove(int index) instead, because this doesn't need a child lookup in the group's table of children. This can be much faster, if there are lots of children.

**12.57.3.27 remove() [2/3]**

```cpp
void Fl_Group::remove (Fl_Widget * o) [inline]
```

Removes the widget o from the group.

**See also**

void remove(Fl_Widget&)

**12.57.3.28 remove() [3/3]**

```cpp
void Fl_Group::remove (int index)
```

Removes the widget at index from the group but does not delete it. This method does nothing if index is out of bounds. This method differs from the clear() method in that it only affects a single widget and does not delete it from memory.

**Since**

FLTK 1.3.0

**12.57.3.29 resizable() [1/3]**

```cpp
Fl_Widget * Fl_Group::resizable ( ) const [inline]
```

Returns the group's resizable widget.

**See** void Fl_Group::resizable(Fl_Widget *o)
**12.57.30 resizable() [2/3]**

void Fl_Group::resizable (  
    Fl_Widget & o ) [inline]

Sets the group's resizable widget.
See void Fl_Group::resizable(Fl_Widget *o)

**12.57.31 resizable() [3/3]**

void Fl_Group::resizable (  
    Fl_Widget * o ) [inline]

The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
If the resizable is NULL the group's size is fixed and all of the widgets in the group remain a fixed size and distance
from the top-left corner. This is the default for groups derived from Fl_Window and Fl_Pack.
The resizable may be set to the group itself, in which case all of the widgets that are its direct children are resized
proportionally. This is the default value for Fl_Group.
The resizable widget defines the resizing box for the group, which could be the group itself or one of the group's
direct children. When the group is resized it calculates a new size and position for all of its children. Widgets that
are horizontally or vertically inside the dimensions of the box are scaled to the new size. Widgets outside the box
are moved.

Note

The resizable of a group **must** be one of

- NULL
- the group itself
- a direct child of the group.

If you set any other widget that is not a direct child of the group as its resizable then the behavior is undefined.
This is **not** checked by Fl_Group for historical reasons.

In these examples the gray area is the resizable:
It is possible to achieve any type of resize behavior by using an invisible \texttt{Fl\_Box} as the resizable and/or by using a hierarchy of \texttt{Fl\_Group} widgets, each with their own resizing strategies. See the \texttt{How Does Resizing Work?} chapter for more examples and detailed explanation.

Note

The \texttt{resizable()} widget of a window can also affect the window's resizing behavior if \texttt{Fl\_Window::size\_range()} is not called. Please see \texttt{Fl\_Window::default\_size\_range()} for more information on how the default size range is calculated.

See also

\texttt{Fl\_Window::size\_range()}
\texttt{Fl\_Window::default\_size\_range()}

\subsubsection{12.57.3.32 resize()}

\begin{verbatim}
void Fl\_Group::resize (int X, int Y, int W, int H) [virtual]
\end{verbatim}

Resizes the \texttt{Fl\_Group} widget and all of its children. The \texttt{Fl\_Group} widget first resizes itself, and then it moves and resizes all its children according to the rules documented for \texttt{Fl\_Group::resizable(Fl\_Widget*)}

See also

\texttt{Fl\_Group::resizable(Fl\_Widget*)}
\texttt{Fl\_Group::resizable()}
\texttt{Fl\_Widget::resize(int,int,int,int)}

Reimplemented from \texttt{Fl\_Widget}.
Reimplemented in \texttt{Fl\_Input\_Choice, Fl\_Pack, Fl\_Scroll, Fl\_Spinner, Fl\_Table, Fl\_Terminal, Fl\_Text\_Display, Fl\_Tile, Fl\_Window, Fl\_Help\_View, Fl\_Overlay\_Window, Fl\_Tabs, and Fl\_Tree}.

\subsubsection{12.57.3.33 sizes()}

\begin{verbatim}
int * Fl\_Group::sizes () [protected]
\end{verbatim}

Returns the internal array of widget sizes and positions. For backward compatibility with FLTK versions before 1.4.

The \texttt{sizes()} array stores the initial positions of widgets as (left, right, top, bottom) quads. The first quad is the group, the second is the resizable (clipped to the group), and the rest are the children. If the group and/or the \texttt{resizable()} is a \texttt{Fl\_Window}, then the first (left) and third (top) entries of their respective quads (x,y) are zero.
**Deprecated**  Deprecated since 1.4.0. Please use `bounds()` instead.

**Note**  This method will be removed in a future FLTK version (1.5.0 or higher).

**Returns**  
Array of int's with widget positions and sizes. The returned array is only valid until `init_sizes()` is called or widgets are added to or removed from the group.

**Note**  Since FLTK 1.4.0 the returned array is a **read-only** and re-ordered copy of the internal `bounds()` array. Do not change its contents. If you need to rearrange children in a group, do so by resizing the children and call `init_sizes()`.

**See also**

`bounds()`

### 12.57.34  update_child()

```cpp
void Fl_Group::update_child ( Fl_Widget & widget ) const  [protected]
```

**Draws a child only if it needs it.**  
This draws a child widget, if it is not clipped and if any `damage()` bits are set. The damage bits are cleared after drawing.

**See also**

`Fl_Group::draw_child(Fl_Widget& widget) const`

The documentation for this class was generated from the following files:

- `Fl_Group.H`
- `Fl_Group.cxx`
- `forms_compatibility.cxx`

### 12.58  Fl_Help_Block Struct Reference

**Public Attributes**

- `Fl_Color bgcolor`
- `uchar border`
- `const char * end`
- `int h`
- `int line [32]`
- `int ol`
- `int ol_num`
- `const char * start`
- `int w`
- `int x`
- `int y`

The documentation for this struct was generated from the following file:

- `Fl_Help_View.H`
12.59 Fl_Help_Dialog Class Reference

The Fl_Help_Dialog widget displays a standard help dialog window using the Fl_Help_View widget.

Public Member Functions

- **Fl_Help_Dialog ()**
  The constructor creates the dialog pictured above.

- **int h ()**
  Returns the position and size of the help dialog.

- **void hide ()**
  Hides the Fl_Help_Dialog window.

- **int load (const char ∗f)**
  Loads the specified HTML file into the Fl_Help_View widget.

- **void position (int xx, int yy)**
  Set the screen position of the dialog.

- **void resize (int xx, int yy, int ww, int hh)**
  Change the position and size of the dialog.

- **void show ()**
  Shows the Fl_Help_Dialog window.

- **void show (int argc, char ∗∗argv)**
  Shows the main Help Dialog Window Delegates call to encapsulated window void Fl_Window::show(int argc, char ∗∗argv) instance method.

- **Fl_Fontsize textsize ()**
  Sets or gets the default text size for the help view.

- **void textsize (Fl_Fontsize s)**
  Sets or gets the default text size for the help view.

- **void topline (const char ∗n)**
  Sets the top line in the Fl_Help_View widget to the named or numbered line.

- **void topline (int n)**
  Sets the top line in the Fl_Help_View widget to the named or numbered line.

- **const char ∗ value () const**
  The first form sets the current buffer to the string provided and reformats the text.

- **void value (const char ∗f)**
  The first form sets the current buffer to the string provided and reformats the text.

- **int visible ()**
  Returns 1 if the Fl_Help_Dialog window is visible.

- **int w ()**
  Returns the position and size of the help dialog.

- **int x ()**
  Returns the position and size of the help dialog.

- **int y ()**
  Returns the position and size of the help dialog.

- **~Fl_Help_Dialog ()**
  The destructor destroys the widget and frees all memory that has been allocated for the current file.
12.59.1 Detailed Description

The Fl_Help_Dialog widget displays a standard help dialog window using the Fl_Help_View widget. The Fl_Help_Dialog class is not part of the FLTK core library, but instead of fltk_images. Use --use-images when compiling with fltk-config.

![Figure 12.23 Fl_Help_Dialog](image)

12.59.2 Member Function Documentation

12.59.2.1 load()

```cpp
int Fl_Help_Dialog::load (const char * f)
```

Loads the specified HTML file into the Fl_Help_View widget. The filename can also contain a target name ("filename.html#target"). Always use forward slashes as path delimiters, MSWindows-style backslashes are not supported here.

**Parameters**

- `f` the name and path of an HTML file

**Returns**

0 on success, -1 on error

**See also**

Fl_Help_View::load(), fl_load_uri()

12.59.2.2 show()

```cpp
void Fl_Help_Dialog::show ( )
```

Shows the Fl_Help_Dialog window. Shows the main Help Dialog Window Delegates call to encapsulated window _ void Fl_Window::show() method.

12.59.2.3 textsize()

```cpp
void Fl_Help_Dialog::textsize (Fl_Fontsize s)
```

Sets or gets the default text size for the help view. Sets the internal Fl_Help_View instance text size. Delegates call to encapsulated view _ void Fl_Help_View::textsize(Fl_Fontsize s) instance method
12.59.2.4 value() [1/2]

const char * Fl_Help_Dialog::value ( ) const
The first form sets the current buffer to the string provided and reformats the text.
It also clears the history of the "back" and "forward" buttons. The second form returns the current buffer contents.

12.59.2.5 value() [2/2]

void Fl_Help_Dialog::value ( const char * v )
The first form sets the current buffer to the string provided and reformats the text.
It also clears the history of the "back" and "forward" buttons. The second form returns the current buffer contents.

The documentation for this class was generated from the following files:

• Fl_Help_Dialog.H
• Fl_Help_Dialog.cxx
• Fl_Help_Dialog_Dox.cxx

12.60 Fl_Help_Font_Stack Struct Reference

Public Member Functions

• size_t count ( ) const
  Gets the current count of font style elements in the stack.

• Fl_Help_Font_Stack ( )
  font stack construction, initialize attributes.

• void init ( Fl_Font f, Fl_Fontsize s, Fl_Color c)

• void pop ( Fl_Font &f, Fl_Fontsize &s, Fl_Color &c)
  Pops from the stack the font style triplet and calls fl_font() & fl_color() adequately.

• void push ( Fl_Font f, Fl_Fontsize s, Fl_Color c)
  Pushes the font style triplet on the stack, also calls fl_font() & fl_color() adequately.

• void top ( Fl_Font &f, Fl_Fontsize &s, Fl_Color &c)
  Gets the top (current) element on the stack.

Protected Attributes

• Fl_Help_Font_Style elts_[MAX_FL_HELP_FS_ELTS]
  font elements

• size_t nfonts_
  current number of fonts in stack

The documentation for this struct was generated from the following file:

• Fl_Help_View.H

12.61 Fl_Help_Font_Style Struct Reference

Fl_Help_View font stack element definition.
#include <Fl_Help_View.H>

Public Member Functions

• Fl_Help_Font_Style ( Fl_Font afont, Fl_Fontsize asize, Fl_Color acolor)
• void get ( Fl_Font &afont, Fl_Fontsize &asize, Fl_Color &acolor)
  Gets current font attributes.

• void set ( Fl_Font afont, Fl_Fontsize asize, Fl_Color acolor)
  Sets current font attributes.
Public Attributes

- **Fl_Color c**
  
  Font Color.

- **Fl_Font f**
  
  Font.

- **Fl_Fontsize s**
  
  Font Size.

12.61.1 Detailed Description

Fl_Help_View font stack element definition.

The documentation for this struct was generated from the following file:

- Fl_Help_View.H

12.62 Fl_Help_Link Struct Reference

Definition of a link for the html viewer.

#include <Fl_Help_View.H>

Public Attributes

- **char filename [192]**
  
  Reference filename.

- **int h**
  
  Height of link text.

- **char name [32]**
  
  Link target (blank if none)

- **int w**
  
  Width of link text.

- **int x**
  
  X offset of link text.

- **int y**
  
  Y offset of link text.

12.62.1 Detailed Description

Definition of a link for the html viewer.

The documentation for this struct was generated from the following file:

- Fl_Help_View.H

12.63 Fl_Help_Target Struct Reference

Fl_Help_Target structure.

#include <Fl_Help_View.H>

Public Attributes

- **char name [32]**
  
  Target name.

- **int y**
  
  Y offset of target.
12.64 Fl_Help_View Class Reference

The Fl_Help_View widget displays HTML text.
#include <Fl_Help_View.H>
Inheritance diagram for Fl_Help_View:

```
Fl_Help_View
`s
Fl_Group `s
Fl_Widget `s
```

Public Member Functions

- void clear_selection ()
  Removes the current text selection.
- const char * directory () const
  Returns the current directory for the text in the buffer.
- const char * filename () const
  Returns the current filename for the text in the buffer.
- int find (const char * s, int p=0)
  Finds the specified string s at starting position p.
- Fl_Help_View (int xx, int yy, int ww, int hh, const char * l=0)
  The constructor creates the Fl_Help_View widget at the specified position and size.
- int handle (int) FL_OVERRIDE
  Handles events in the widget.
- int leftline () const
  Gets the left position in pixels.
- void leftline (int)
  Scrolls the text to the indicated position, given a pixel column.
- void link (Fl_Help_Func *fn)
  This method assigns a callback function to use when a link is followed or a file is loaded (via Fl_Help_View::load()) that requires a different file or path.
- int load (const char * f)
  Loads the specified file.
- void resize (int, int, int, int) FL_OVERRIDE
  Resizes the help widget.
- int scrollbar_size () const
  Gets the current size of the scrollbars’ troughs, in pixels.
- void scrollbar_size (int newSize)
  Sets the pixel size of the scrollbars’ troughs to newSize, in pixels.
- void select_all ()
  Selects all the text in the view.
- int size () const
Gets the size of the help view.

- **void** `size` ([int W], [int H])
  
  Returns the current default text color.

- **void** `textcolor` ([Fl_Color c])
  
  Sets the default text color.

- **Fl_Color** `textcolor` () const
  
  Returns the current default text color.

- **void** `textcolor` ([Fl_Color c])
  
  Sets the default text color.

- **Fl_Font** `textfont` () const
  
  Returns the current default text font.

- **void** `textfont` ([Fl_Font f])
  
  Sets the default text font.

- **Fl_Font** `textfont` () const
  
  Returns the current default text font.

- **void** `textfont` ([Fl_Font f])
  
  Sets the default text font.

- **Fl_Fontsize** `textsize` () const
  
  Gets the default text size.

- **void** `textsize` ([Fl_Fontsize s])
  
  Sets the default text size.

- **const char** `∗` `title` ()
  
  Returns the current document title, or NULL if there is no title.

- **int** `topline` () const
  
  Returns the current top line in pixels.

- **void** `topline` ([const char ∗n])
  
  Scrolls the text to the indicated position, given a named destination.

- **void** `topline` ([int])
  
  Scrolls the text to the indicated position, given a pixel line.

- **const char** `∗` `value` () const
  
  Returns the current buffer contents.

- **void** `value` ([const char ∗val])
  
  Sets the current help text buffer to the string provided and reformats the text.

- **~Fl_Help_View** ()
  
  Destroys the Fl_Help_View widget.

### Public Member Functions inherited from Fl_Group

- **Fl_Widget ∗& _ddfdesign_kludge** ()
  
  This is for forms compatibility only.

- **void** `add` ([Fl_Widget &])
  
  The widget is removed from its current group (if any) and then added to the end of this group.

- **void** `add` ([Fl_Widget ∗o])
  
  See void Fl_Group::add(Fl_Widget &w)

- **void** `add_resizable` ([Fl_Widget ∗&o])
  
  Adds a widget to the group and makes it the resizable widget.

- **Fl_Widget ∗const ∗ array** () const
  
  Returns a pointer to the array of children.

- **Fl_Group const ∗ as_group** () const FL_OVERRIDE
  
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- **void** `begin` ()
  
  Sets the current group so you can build the widget tree by just constructing the widgets.

- **Fl_Widget ∗ child** ([int n]) const
  
  Returns `array()[n].`

- **int** `children` () const
  
  Returns how many child widgets the group has.

- **void** `clear` ()

Generated by Doxygen
12.64 Fl_Help_View Class Reference

Deletes all child widgets from memory recursively.

- **unsigned int clip_children ()**
  - Returns the current clipping mode.
- **virtual void clip_children (int c)**
  - Controls whether the group widget clips the drawing of child widgets to its bounding box.
- **virtual int delete_child (int n)**
  - Removes the widget at index from the group and deletes it.
- **void end ()**
  - Exactly the same as current(this->parent()).
- **int find (const Fl_Widget &o) const**
  - See int Fl_Group::find(const Fl_Widget *w) const.
- **int find (const Fl_Widget *) const**
  - Searches the child array for the widget and returns the index.
- **Fl_Group (int, int, int, int, const char * = 0)**
  - Creates a new Fl_Group widget using the given position, size, and label string.
- **void focus (Fl_Widget *)**
- **void forms_end ()**
  - This is for forms compatibility only.
- **void init_sizes ()**
  - Resets the internal array of widget sizes and positions.
- **void insert (Fl_Widget &, int i)**
  - The widget is removed from its current group (if any) and then inserted into this group.
- **void insert (Fl_Widget &o, Fl_Widget *)**
  - This does insert(w, find(before)).
- **void remove (Fl_Widget &)**
  - Removes a widget from the group but does not delete it.
- **void remove (Fl_Widget *o)**
  - Removes the widget o from the group.
- **void remove (int index)**
  - Removes the widget at index from the group but does not delete it.
- **Fl_Widget * resizable () const**
  - Returns the group's resizable widget.
- **void resizable (Fl_Widget &o)**
  - Sets the group's resizable widget.
- **void resizable (Fl_Widget *)**
  - The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- **virtual ~Fl_Group ()**
  - The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- **void _clear_fullscreen ()**
- **void _set_fullscreen ()**
- **void activate ()**
  - Activates the widget.
- **unsigned int active () const**
  - Returns whether the widget is active.
- **int active_r () const**
  - Returns whether the widget and all of its parents are active.
- **Fl_Align align () const**
  - Gets the label alignment.

Generated by Doxygen
- void align (Fl_Align alignment)
  
  Sets the label alignment.
- long argument () const
  
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ as_gl_window () const
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual Fl_Window ∗ as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ as_window () const
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- void bind_deimage (Fl_Image ∗ img)
  
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗ img)
  
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  
  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb)
  
  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗ cb, void ∗ p)
  
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback ∗ cb)
  
  Sets the current callback function for the widget.
- void callback (Fl_Callback0 ∗ cb)
  
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗ cb, long p=0)
  
  Sets the current callback function for the widget.
- unsigned int changed () const
  
  Checks if the widget value changed since the last callback.
- void clear_active ()
  
  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()
  
  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)
  
  Clears or sets the damage flags.
- void clear_output ()
  
  Sets a widget to accept input.
- void clear_visible ()
  
  Hides the widget.
- void clear_visible_focus ()
  
  Disables keyboard focus navigation with this widget.
- Fl_Color color () const
Gets the background color of the widget.

• void color (Fl_Color bg)

Sets the background color of the widget.

• void color (Fl_Color bg, Fl_Color sel)

Sets the background and selection color of the widget.

• Fl_Color color2 () const

For back compatibility only.

• void color2 (unsigned a)

For back compatibility only.

• int contains (const Fl_Widget *w) const

Checks if w is a child of this widget.

• void copy_label (const char *new_label)

Sets the current label.

• void copy_tooltip (const char *text)

Sets the current tooltip text.

• uchar damage () const

Returns non-zero if draw() needs to be called.

• void damage (uchar c)

Sets the damage bits for the widget.

• void damage (uchar c, int x, int y, int w, int h)

Sets the damage bits for an area inside the widget.

• int damage_resize (int, int, int, int)

Internal use only.

• void deactivate ()

Deactivates the widget.

• Fl_Image * deimage ()

Gets the image that is used as part of the widget label when in the inactive state.

• const Fl_Image * deimage () const

Gets the image that is used as part of the widget label when in the inactive state.

• void deimage (Fl_Image &img)

Sets the image to use as part of the widget label when in the inactive state.

• void deimage (Fl_Image *img)

Sets the image to use as part of the widget label when in the inactive state.

• int deimage_bound () const

Returns whether the inactive image is managed by the widget.

• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

Calls the widget callback function with default arguments.

• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

Calls the widget callback function with arbitrary arguments.

• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

Calls the widget callback function with arbitrary arguments.

• void draw_label (int, int, int, int, Fl_Align) const

Draws the label in an arbitrary bounding box with an arbitrary alignment.

• int h () const

Gets the widget height.

• virtual void hide ()

Makes a widget invisible.

• Fl_Image * image ()

Gets the image that is used as part of the widget label when in the active state.

• const Fl_Image * image () const

Gets the image that is used as part of the widget label when in the active state.
- **void image (Fl_Image &img)**
  Sets the image to use as part of the widget label when in the active state.
- **void image (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the active state.
- **int image_bound () const**
  Returns whether the image is managed by the widget.
- **int inside (const Fl_Widget *wgt) const**
  Checks if this widget is a child of wgt.
- **int is_label_copied () const**
  Returns whether the current label was assigned with copy_label().
- **const char * label () const**
  Gets the current label text.
- **void label (const char *text)**
  Sets the current label pointer.
- **void label (Fl_Labeltype a, const char *b)**
  Shortcut to set the label text and type in one call.
- **Fl_Color labelcolor () const**
  Gets the label color.
- **void labelcolor (Fl_Color c)**
  Sets the label color.
- **Fl_Font labelfont () const**
  Gets the font to use.
- **void labelfont (Fl_Font f)**
  Sets the font to use.
- **Fl_Fontsize labelsize () const**
  Gets the font size in pixels.
- **void labelsize (Fl_Fontsize pix)**
  Sets the font size in pixels.
- **Fl_Labeltype labeltype () const**
  Gets the label type.
- **void labeltype (Fl_Labeltype a)**
  Sets the label type.
- **void measure_label (int &ww, int &hh) const**
  Sets width ww and height hh accordingly with the label size.
- **bool needs_keyboard () const**
  Returns whether this widget needs a keyboard.
- **void needs_keyboard (bool needs)**
  Sets whether this widget needs a keyboard.
- **unsigned int output () const**
  Returns if a widget is used for output only.
- **Fl_Group * parent () const**
  Returns a pointer to the parent widget.
- **void parent (Fl_Group *p)**
  Internal use only - "for hacks only".
- **void position (int X, int Y)**
  Repositions the window or widget.
- **void redraw ()**
  Schedules the drawing of the widget.
- **void redraw_label ()**
  Schedules the drawing of the label.
- **Fl_Color selection_color () const**
Gets the selection color.

- void selection_color (Fl_Color a)
  Sets the selection color.

- void set_active ()
  Marks the widget as active without sending events or changing focus.

- void set_changed ()
  Marks the value of the widget as changed.

- void set_output ()
  Sets a widget to output only.

- void set_visible ()
  Makes the widget visible.

- void set_visible_focus ()
  Enables keyboard focus navigation with this widget.

- int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.

- void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

- virtual void show ()
  Makes a widget visible.

- void size (int W, int H)
  Changes the size of the widget.

- int take_focus ()
  Gives the widget the keyboard focus.

- unsigned int takesevents () const
  Returns if the widget is able to take events.

- int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.

- const char *tooltip () const
  Gets the current tooltip text.

- void tooltip (const char *text)
  Sets the current tooltip text.

- Fl_Window *top_window () const
  Returns a pointer to the top-level window for the widget.

- Fl_Window *top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const
  Gets the widget type.

- void type (uchar t)
  Sets the widget type.

- int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void *user_data () const
  Gets the user data for this widget.

- void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

- void user_data (void *v)
  Sets the user data for this widget.

- unsigned int visible () const
  Returns whether a widget is visible.

- unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  
  
  Modifies keyboard focus navigation.
• int visible_r () const
  
  Returns whether a widget and all its parents are visible.
• int w () const
  
  Gets the widget width.
• Fl_When when () const
  
  Returns the conditions under which the callback is called.
• void when (uchar i)
  
  Sets the flags used to decide when a callback is called.
• Fl_Window ∗ window () const
  
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  
  Gets the widget position in its window.
• int y () const
  
  Gets the widget position in its window.
• virtual ∼Fl_Widget ()
  
  Destroys the widget.

Protected Member Functions

• void draw () FL_OVERRIDE
  
  Draws the Fl_Help_View widget.

Protected Member Functions inherited from Fl_Group

• Fl_Rect ∗ bounds ()
  
  Returns the internal array of widget sizes and positions.
• void draw_child (Fl_Widget &widget) const
  
  Forces a child to redraw.
• void draw_children ()
  
  Draws all children of the group.
• void draw_outside_label (const Fl_Widget &widget) const
  
  Parents normally call this to draw outside labels of child widgets.
• virtual int on_insert (Fl_Widget ∗, int)
  
  Allow derived groups to act when a widget is added as a child.
• virtual int on_move (int, int)
  
  Allow derived groups to act when a widget is moved within the group.
• virtual void on_remove (int)
  
  Allow derived groups to act when a child widget is removed from the group.
• int ∗ sizes ()
  
  Returns the internal array of widget sizes and positions.
• void update_child (Fl_Widget &widget) const
  
  Draws a child only if it needs it.
Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y , int W, int H) const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void draw_label () const
  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.
- unsigned int flags () const
  Gets the widget flags mask.
- void h (int v)
  Internal use only.
- void set_flag (unsigned int c)
  Sets a flag in the flags mask.
- void w (int v)
  Internal use only.
- void x (int v)
  Internal use only.
- void y (int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group ∗ current ()
  Returns the currently active group.
- static void current (Fl_Group ∗g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)
  The default callback for all widgets that don’t set a callback.
- static unsigned int labelShortcut (const char ∗t)
  Returns the Unicode value of the ‘&x’ shortcut in a given text.
- static int testShortcut (const char ∗, const bool require_alt=false)
  Returns true if the given text t contains the entered ‘&x’ shortcut.
Protected Types inherited from Fl_Widget

\[
\begin{align*}
\text{enum } & \{ \\
\text{INACTIVE} = 1<<0, \text{INVISIBLE} = 1<<1, \text{OUTPUT} = 1<<2, \text{NOBORDER} = 1<<3, \\
\text{FORCE\_POSITION} = 1<<4, \text{NON\_MODAL} = 1<<5, \text{SHORTCUT\_LABEL} = 1<<6, \text{CHANGED} = 1<<7, \\
\text{OVERRIDE} = 1<<8, \text{VISIBLE\_FOCUS} = 1<<9, \text{COPIED\_LABEL} = 1<<10, \text{CLIP\_CHILDREN} = 1<<11, \\
\text{MENU\_WINDOW} = 1<<12, \text{TOOLTIP\_WINDOW} = 1<<13, \text{MODAL} = 1<<14, \text{NO\_OVERLAY} = 1<<15, \\
\text{GROUP\_RELATIVE} = 1<<16, \text{COPIED\_TOOLTIP} = 1<<17, \text{FULLSCREEN} = 1<<18, \text{MAC\_USE\_ACCENTS\_MENU} = 1<<19, \\
\text{NEEDS\_KEYBOARD} = 1<<20, \text{IMAGE\_BOUND} = 1<<21, \text{DEIMAGE\_BOUND} = 1<<22, \\
\text{AUTO\_DELETE\_USER\_DATA} = 1<<23, \text{MAXIMIZED} = 1<<24, \text{USERFLAG3} = 1<<29, \text{USERFLAG2} = 1<<30, \text{USERFLAG1} = 1<<31 \} \\
\end{align*}
\]
flags possible values enumeration.

12.64.1 Detailed Description

The Fl_Help_View widget displays HTML text. Most HTML 2.0 elements are supported, as well as a primitive implementation of tables. GIF, JPEG, and PNG images are displayed inline. Supported HTML tags:

- A: HREF/NAME
- B
- BODY: BGCOLOR/TEXT/LINK
- BR
- CENTER
- CODE
- DD
- DL
- DT
- EM
- FONT: COLOR/SIZE/FACE=(helvetica/arial/sans/times/serif/symbol/courier)
- H1/H2/H3/H4/H5/H6
- HEAD
- HR
- I
- IMG: SRC/ WIDTH/HEIGHT/ALT
- KBD
- LI
- OL
- P
- PRE

Generated by Doxygen
• STRONG
• TABLE: TH/TD/TR/BORDER/BGCOLOR/COLSPAN/ALIGN=CENTER|RIGHT|LEFT
• TITLE
• TT
• U
• UL
• VAR

Supported color names:
  • black, red, green, yellow, blue, magenta, fuchsia, cyan, aqua, white, gray, grey, lime, maroon, navy, olive, purple, silver, teal.

Supported urls:
  • Internal: file:
  • External: http: ftp: https: ipp: mailto: news:

Quoted char names:
  • Aacute aacute Acirc acirc acute AElig aelig Agrave agrave amp Aring aring Atilde atilde Auml auml
  • brvbar bull
  • Ccedil ccedil cedil cent copy curren
  • dagger deg divide
  • Eacute eacute Ecirc ecirc Egrave egrave ETH eth Euml euml euro
  • frac12 frac14 frac34
  • gt
  • Iacute iacute Icirc icirc iexcl Igrave igrave iquest Iuml iuml
  • laquo lt
  • macr micro middot
  • nbsp not Ntilde ntilde
  • Oacute oacute Ocirc ocirc Ograve ograve ordf ordm Oslash oslash Otilde otilde Ouml ouml
  • para permil plusmn pound
  • quot
  • raquo reg
  • sect shy sup1 sup2 sup3 szlig
  • THORN thorn times trade
  • Uacute uacute Ucirc ucirc Ugrave ugrave uml Uuml uuml
  • Yacute yacute
  • yen Yuml yuml

Note
You can't effectively set the box() to FL_NO_BOX, this would result in FL_DOWN_BOX being used as the boxtype of the widget. This is unexpected but can't be changed for backwards compatibility. If you don't want a frame around the widget you can use FL_FLAT_BOX instead.
12.64.2 Constructor & Destructor Documentation

12.64.2.1 ∼Fl_Help_View()

Fl_Help_View::~Fl_Help_View ()

Destroys the Fl_Help_View widget.
The destructor destroys the widget and frees all memory that has been allocated for the current document.

12.64.3 Member Function Documentation

12.64.3.1 draw()

void Fl_Help_View::draw (void) [protected], [virtual]

Draws the Fl_Help_View widget.
Reimplemented from Fl_Group.

12.64.3.2 find()

int Fl_Help_View::find (const char * s, int p = 0)

Finds the specified string s at starting position p.
The argument p and the return value are offsets in Fl_Help_View::value(), counting from 0. If p is out of range, 0 is used.
The string comparison is simple but honors some special cases:

- the specified string s must be in UTF-8 encoding
- HTML tags in value() are filtered (not compared as such, they never match)
- HTML entities like '&lt;' or '&x#20ac;' are converted to Unicode (UTF-8)
- ASCII characters (7-bit, < 0x80) are compared case insensitive
- every newline (LF, \n) in value() is treated like a single space
- all other strings are compared as-is (byte by byte)

Todo complex HTML entities for Unicode code points > 0x80 are currently treated like one byte (not character!) and do not (yet) match correctly ("<" matches "<" but "€" doesn't match "€", and "Ü" doesn't match "Ü")

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>s</th>
<th>search string in UTF-8 encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>p</td>
<td>starting position for search (0,...), Default = 0</td>
</tr>
</tbody>
</table>

Returns

the matching position or -1 if not found

12.64.3.3 handle()

int Fl_Help_View::handle (int event) [virtual]

Handles events in the widget.
Reimplemented from Fl_Group.
12.64.3.4 leftline()

```cpp
void Fl_Help_View::leftline ( int left )
```

Scrolls the text to the indicated position, given a pixel column. If the given pixel value `left` is out of range, then the text is scrolled to the left or right side of the document, resp.

**Parameters**

| in | `left` | left column number in pixels (0 = left side) |

12.64.3.5 link()

```cpp
void Fl_Help_View::link ( Fl_Help_Func * fn ) [inline]
```

This method assigns a callback function to use when a link is followed or a file is loaded (via `Fl_Help_View::load()`) that requires a different file or path. The callback function receives a pointer to the `Fl_Help_View` widget and the URI or full pathname for the file in question. It must return a pathname that can be opened as a local file or NULL:

```cpp
const char * fn(Fl_Widget *w, const char *uri);
```

The link function can be used to retrieve remote or virtual documents, returning a temporary file that contains the actual data. If the link function returns NULL, the value of the `Fl_Help_View` widget will remain unchanged. If the link callback cannot handle the URI scheme, it should return the uri value unchanged or set the `value()` of the widget before returning NULL.

12.64.3.6 load()

```cpp
int Fl_Help_View::load ( const char * f )
```

Loads the specified file. This method loads the specified file or URL. The filename may end in a `#name` style target. If the URL starts with `ftp, http, https, ipp, mailto, or news`, followed by a colon, FLTK will use `fl_open_uri()` to show the requested page in an external browser. In all other cases, the URL is interpreted as a filename. The file is read and displayed in this browser. Note that MSWindows style backslashes are not supported in the file name.

**Parameters**

| in | `f` | filename or URL |

**Returns**

0 on success, -1 on error

**See also**

`fl_open_uri()`

12.64.3.7 resize()

```cpp
void Fl_Help_View::resize ( int xx, int yy, int ww, int hh ) [virtual]
```

Resizes the help widget. Reimplemented from `Fl_Group`. 

Generated by Doxygen
### scrollbar_size() [1/2]

```cpp
int Fl_Help_View::scrollbar_size () const [inline]
```

Gets the current size of the scrollbars' troughs, in pixels. If this value is zero (default), this widget will use the Fl:scrollbar_size() value as the scrollbar's width. Returns

Scrollbar size in pixels, or 0 if the global Fl:scrollbar_size() is being used.

**See also**

Fl:scrollbar_size(int)

### scrollbar_size() [2/2]

```cpp
void Fl_Help_View::scrollbar_size (int newSize) [inline]
```

Sets the pixel size of the scrollbars' troughs to newSize, in pixels. Normally you should not need this method, and should use Fl:scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want. Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare. Setting newSize to the special value of 0 causes the widget to track the global Fl:scrollbar_size(), which is the default.

**Parameters**

| in  | newSize | Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl:scrollbar_size() |

**See also**

Fl:scrollbar_size()

### topline() [1/2]

```cpp
void Fl_Help_View::topline (const char * n )
```

Scrolls the text to the indicated position, given a named destination.

**Parameters**

| in  | n | target name |

### topline() [2/2]

```cpp
void Fl_Help_View::topline (int top )
```

Scrolls the text to the indicated position, given a pixel line. If the given pixel value top is out of range, then the text is scrolled to the top or bottom of the document, resp.

**Parameters**

| in  | top | top line number in pixels (0 = start of document) |
12.64.3.12 value()

void Fl_Help_View::value (const char ∗val)
Sets the current help text buffer to the string provided and reformats the text.
The provided character string val is copied internally and will be freed when value() is called again, or when the
widget is destroyed.
If val is NULL, then the widget is cleared.
The documentation for this class was generated from the following files:

• Fl_Help_View.H
• Fl_Help_View.cxx

12.65 Fl_Hold_Browser Class Reference

The Fl_Hold_Browser is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking
on the empty space.
#include <Fl_Hold_Browser.H>
Inheritance diagram for Fl_Hold_Browser:

<table>
<thead>
<tr>
<th>Fl_Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Group</td>
</tr>
<tr>
<td>Fl_Browser_</td>
</tr>
<tr>
<td>Fl_Browser</td>
</tr>
<tr>
<td>Fl_Hold_Browser</td>
</tr>
</tbody>
</table>

Public Member Functions

• Fl_Hold_Browser (int X, int Y, int W, int H, const char ∗L=0)
  Creates a new Fl_Hold_Browser widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Browser

• void add (const char ∗newtext, void ∗d=0)
  Adds a new line to the end of the browser.
• void bottomline (int line)
  Scrolls the browser so the bottom item in the browser is showing the specified line.
• void clear ()
  Removes all the lines in the browser.
• char column_char () const
  Gets the current column separator character.
• void column_char (char c)
  Sets the column separator to c.
• const int ∗column_widths () const
  Gets the current column width array.
• void column_widths (const int ∗arr)
  Sets the current array to arr.
• void ∗data (int line) const
Returns the user data() for specified line.

- void data (int line, void *d)
  Sets the user data for specified line to d.

- void display (int line, int val=1)
  For back compatibility.

- int displayed (int line) const
  Returns non-zero if line has been scrolled to a position where it is being displayed.

- Fl_Browser (int X, int Y, int W, int H, const char *L=0)
  The constructor makes an empty browser.

- char format_char () const
  Gets the current format code prefix character, which by default is '@'.

- void format_char (char c)
  Sets the current format code prefix character to c.

- void hide () FL_OVERRIDE
  Hides the entire Fl_Browser widget – opposite of show().

- void hide (int line)
  Makes line invisible, preventing selection by the user.

- Fl_Image * icon (int line) const
  Returns the icon currently defined for line.

- void icon (int line, Fl_Image *icon)
  Set the image icon for line to the value icon.

- void insert (int line, const char *newtext, void *d=0)
  Insert a new entry whose label is newtext above given line, optional data d.

- void lineposition (int line, Fl_Line_Position pos)
  Updates the browser so that line is shown at position pos.

- int load (const char *filename)
  Clears the browser and reads the file, adding each line from the file to the browser.

- void make_visible (int line)
  Make the item at the specified line visible().

- void middleline (int line)
  Scrolls the browser so the middle item in the browser is showing the specified line.

- void move (int to, int from)
  Line from is removed and reinserted at to.

- void remove (int line)
  Remove entry for given line number, making the browser one line shorter.

- void remove_icon (int line)
  Removes the icon for line.

- void replace (int a, const char *b)
  For back compatibility only.

- int select (int line, int val=1)
  Sets the selection state of the item at line to the value val.

- int selected (int line) const
  Returns 1 if specified line is selected, 0 if not.

- void show () FL_OVERRIDE
  Shows the entire Fl_Browser widget – opposite of hide().

- void show (int line)
  Makes line visible, and available for selection by user.

- int size () const
  Returns how many lines are in the browser.

- void size (int W, int H)

- void swap (int a, int b)
Swaps two browser lines a and b.

- `const char* text(int line) const`
  - Returns the label text for the specified line.
- `void text(int line, const char* newtext)`
  - Sets the text for the specified line to newtext.
- `Fl_Fontsize textsize() const`
  - Gets the default text size (in pixels) for the lines in the browser.
- `void textsize(Fl_Fontsize newSize)`
  - Sets the default text size (in pixels) for the lines in the browser to newSize.
- `int topline() const`
  - Returns the line that is currently visible at the top of the browser.
- `void topline(int line)`
  - Scrolls the browser so the top item in the browser is showing the specified line.
- `int value() const`
  - Returns the line number of the currently selected line, or 0 if none selected.
- `void value(int line)`
  - Sets the browser’s value(), which selects the specified line.
- `int visible(int line) const`
  - Returns non-zero if the specified line is visible, 0 if hidden.

The destructor deletes all list items and destroys the browser.

Public Member Functions inherited from Fl_Browser

- `int deselect(int docallbacks=0)`
  - Deselects all items in the list and returns 1 if the state changed or 0 if it did not.
- `void display(void* item)`
  - Displays the item, scrolling the list as necessary.
- `int handle(int event) FL_OVERRIDE`
  - Handles the event within the normal widget bounding box.
- `uchar has_scrollbar() const`
  - Returns the current scrollbar mode, see Fl_Browser::has_scrollbar(uchar)
- `void has_scrollbar(uchar mode)`
  - Sets whether the widget should have scrollbars or not (default Fl_Browser::BOTH).
- `int hposition() const`
  - Gets the horizontal scroll position of the list as a pixel position.
- `void hposition(int)`
  - Sets the horizontal scroll position of the list to pixel position.
- `int linespacing() const`
  - Return the height of additional spacing between browser lines.
- `void linespacing(int pixels)`
  - Add some space between browser lines.
- `int position() const`
  - Get the position.
- `void position(int pos)`
  - Position.
- `void position(int x, int y)`
  - Position.
- `void resize(int X, int Y, int W, int H) FL_OVERRIDE`
  - Repositions and/or resizes the browser.
- `void scrollbar_left()`
  - Moves the vertical scrollbar to the lefthand side of the list.
- `void scrollbar_right()`
  - Moves the vertical scrollbar to the righthand side of the list.
• int scrollbar_size () const
  Gets the current size of the scrollbars' troughs, in pixels.
• void scrollbar_size (int newSize)
  Sets the pixel size of the scrollbars' troughs to newSize, in pixels.
• int scrollbar_width () const
  Returns the global value Fl::scrollbar_size().
• void scrollbar_width (int width)
  Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.
• int select (void ∗item, int val=1, int docallbacks=0)
  Sets the selection state of item to val, and returns 1 if the state changed or 0 if it did not.
• int select_only (void ∗item, int docallbacks=0)
  Selects item and returns 1 if the state changed or 0 if it did not.
• void sort (int flags=0)
  Sorts the items in the browser based on flags.
• Fl_Color textcolor () const
  Gets the default text color for the lines in the browser.
• void textcolor (Fl_Color col)
  Sets the default text color for the lines in the browser to color col.
• Fl_Font textfont () const
  Gets the default text font for the lines in the browser.
• void textfont (Fl_Font font)
  Sets the default text font for the lines in the browser to font.
• Fl_Fontsize textsize () const
  Gets the default text size (in pixels) for the lines in the browser.
• void textsize (Fl_Fontsize newSize)
  Sets the default text size (in pixels) for the lines in the browser to size.
• int vposition () const
  Gets the vertical scroll position of the list as a pixel position pos.
• void vposition (int pos)
  Sets the vertical scroll position of the list to pixel position pos.

Public Member Functions inherited from Fl_Group

• FL_Widget ∗&_dddesign_kludge ()
  This is for forms compatibility only.
• void add (FL_Widget ∗)
  The widget is removed from its current group (if any) and then added to the end of this group.
• void add (FL_Widget &w)
  See void FL_Group::add(FL_Widget &w)
• void add_resizable (FL_Widget ∗w)
  Adds a widget to the group and makes it the resizable widget.
• FL_Widget ∗const ∗array () const
  Returns a pointer to the array of children.
• FL_Group const ∗as_group () const FL_OVERRIDE
  Returns an FL_Group pointer if this widget is an FL_Group.
• void begin ()
  Sets the current group so you can build the widget tree by just constructing the widgets.
• FL_Widget ∗child (int n) const
  Returns array()[n].
• int children () const
Returns how many child widgets the group has.

- void clear ()
  Deletes all child widgets from memory recursively.

- unsigned int clip_children ()
  Returns the current clipping mode.

- void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.

- void end ()
  Exactly the same as current(this->parent()).

- int find (const Fl_Widget &o) const
  See int Fl_Group::find(const Fl_Widget *w) const.

- int find (const Fl_Widget *) const
  Searches the child array for the widget and returns the index.

- Fl_Group (int, int, int, int, const char * = 0)
  Creates a new Fl_Group widget using the given position, size, and label string.

- void focus (Fl_Widget *W)
- void forms_end ()
  This is for forms compatibility only.

- int handle (int) FL_OVERRIDE
  Handles the specified event.

- void init_sizes ()
  Resets the internal array of widget sizes and positions.

- void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.

- void insert (Fl_Widget &o, Fl_Widget *before)
  This does insert(w, find(before)).

- void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.

- void remove (Fl_Widget *o)
  Removes the widget o from the group.

- void remove (int index)
  Removes the widget at index from the group but does not delete it.

- Fl_Widget * resizable () const
  Returns the group’s resizable widget.

- void resizable (Fl_Widget &o)
  Sets the group’s resizable widget.

- void resizable (Fl_Widget *o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- void resize (int, int, int, int) FL_OVERRIDE
  Resizes the Fl_Group widget and all of its children.

- virtual ~Fl_Group ()
  The destructor also deletes all the children.
Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  
  Activates the widget.

• unsigned int active () const
  
  Returns whether the widget is active.

• int active_r () const
  
  Returns whether the widget and all of its parents are active.

• Fl_Align align () const
  
  Gets the label alignment.

• void align (Fl_Align alignment)
  
  Sets the label alignment.

• long argument () const
  
  Gets the current user data (long) argument that is passed to the callback function.

• void argument (long v)
  
  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window ∗ as_gl_window ()
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const ∗ as_gl_window () const
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual Fl_Window ∗ as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window const ∗ as_window () const
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• void bind_deimage (Fl_Image ∗ img)
  
  Sets the image to use as part of the widget label when in the inactive state.

• void bind_deimage (int f)
  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

• void bind_image (Fl_Image ∗ img)
  
  Sets the image to use as part of the widget label when in the active state.

• void bind_image (int f)
  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

• Fl_Boxtype box () const
  
  Gets the box type of the widget.

• void box (Fl_Boxtype new_box)
  
  Sets the box type for the widget.

• Fl_Callback_p callback () const
  
  Gets the current callback function for the widget.

• void callback (Fl_Callback ∗ cb)
  
  Sets the current callback function for the widget.

• void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  
  Sets the current callback function and managed user data for the widget.

• void callback (Fl_Callback ∗ cb, void ∗ p)
  
  Sets the current callback function and data for the widget.

• void callback (Fl_Callback0 ∗ cb)
  
  Sets the current callback function for the widget.

• void callback (Fl_Callback1 ∗ cb, long p=0)
  
  Sets the current callback function for the widget.

• unsigned int changed () const
  
  Checks if the widget value changed since the last callback.

• void clear_active ()
Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)
  Clears or sets the damage flags.
- void clear_output ()
  Sets a widget to accept input.
- void clear_visible ()
  Hides the widget.
- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
- Fl_Color color () const
  Gets the background color of the widget.
- void color (Fl_Color bg)
  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
- Fl_Color color2 () const
  For back compatibility only.
- void color2 (unsigned a)
  For back compatibility only.
- int contains (const Fl_Widget ∗ w) const
  Checks if w is a child of this widget.
- void copy_label (const char ∗ new_label)
  Sets the current label.
- void copy_tooltip (const char ∗ text)
  Sets the current tooltip text.
- uchar damage () const
  Returns non-zero if draw() needs to be called.
- void damage (uchar c)
  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
- int damage_resize (int, int, int, int)
  Internal use only.
- void deactivate ()
  Deactivates the widget.
- Fl_Image ∗ deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image ∗ deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image & img)
  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget ∗ widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• **void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  
  Calls the widget callback function with arbitrary arguments.

• **void draw_label (int, int, int, int, Fl_Align) const**
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• **int h () const**
  
  Gets the widget height.

• **Fl_Image * image ()**
  
  Gets the image that is used as part of the widget label when in the active state.

• **const Fl_Image * image () const**
  
  Gets the image that is used as part of the widget label when in the active state.

• **void image (Fl_Image &img)**
  
  Sets the image to use as part of the widget label when in the active state.

• **void image (Fl_Image *img)**
  
  Sets the image to use as part of the widget label when in the active state.

• **int image_bound () const**
  
  Returns whether the image is managed by the widget.

• **int inside (const Fl_Widget *wgt) const**
  
  Checks if this widget is a child of wgt.

• **int is_label_copied () const**
  
  Returns whether the current label was assigned with copy_label().

• **const char * label () const**
  
  Gets the current label text.

• **void label (const char *text)**
  
  Sets the current label pointer.

• **void label (Fl_Labeltype a, const char *b)**
  
  Shortcut to set the label text and type in one call.

• **Fl_Color labelcolor () const**
  
  Gets the label color.

• **void labelcolor (Fl_Color c)**
  
  Sets the label color.

• **Fl_Font labelfont () const**
  
  Gets the font to use.

• **void labelfont (Fl_Font f)**
  
  Sets the font to use.

• **Fl_Fontsize labelsize () const**
  
  Gets the font size in pixels.

• **void labelsize (Fl_Fontsize pix)**
  
  Sets the font size in pixels.

• **Fl_Labeltype labeltype () const**
  
  Gets the label type.

• **void labeltype (Fl_Labeltype a)**
  
  Sets the label type.

• **void measure_label (int &ww, int &hh) const**
  
  Sets width ww and height hh accordingly with the label size.

• **bool needs_keyboard () const**
  
  Returns whether this widget needs a keyboard.

• **void needs_keyboard (bool needs)**
  
  Sets whether this widget needs a keyboard.

• **unsigned int output () const**
  
  Returns if a widget is used for output only.
Returns a pointer to the parent widget.

- void parent (FL_Group *p)
  
  Internal use only - "for hacks only".

- void position (int X, int Y)
  
  Repositions the window or widget.

- void redraw()
  
  Schedules the drawing of the widget.

- void redraw_label()
  
  Schedules the drawing of the label.

- FL_Color selection_color () const
  
  Gets the selection color.

- void selection_color (FL_Color a)
  
  Sets the selection color.

- void set_active()
  
  Marks the widget as active without sending events or changing focus.

- void set_changed()
  
  Marks the value of the widget as changed.

- void set_output()
  
  Sets a widget to output only.

- void set_visible()
  
  Makes the widget visible.

- void set_visible_focus()
  
  Enables keyboard focus navigation with this widget.

- int shortcut_label () const
  
  Returns whether the widget's label uses '&' to indicate shortcuts.

- void shortcut_label (int value)
  
  Sets whether the widget's label uses '&' to indicate shortcuts.

- void size (int W, int H)
  
  Changes the size of the widget.

- int take_focus()
  
  Gives the widget the keyboard focus.

- unsigned int takesevents () const
  
  Returns if the widget is able to take events.

- int test_shortcut()
  
  Returns true if the widget's label contains the entered '&x' shortcut.

- const char * tooltip () const
  
  Gets the current tooltip text.

- void tooltip (const char *text)
  
  Sets the current tooltip text.

- FL_Window * top_window () const
  
  Returns a pointer to the top-level window for the widget.

- FL_Window * top_window_offset (int &xoff, int &yoff) const
  
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const
  
  Gets the widget type.

- void type (uchar t)
  
  Sets the widget type.

- int use_accents_menu ()
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void * user_data () const
  
  Gets the user data for this widget.
• void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)
  
  Sets the user data for this widget.

• void **user_data** (void ∗v)
  
  Sets the user data for this widget.

• unsigned int **visible () const**
  
  Returns whether a widget is visible.

• unsigned int **visible_focus () const**
  
  Checks whether this widget has a visible focus.

• void **visible_focus (int v)**
  
  Modifies keyboard focus navigation.

• int **visible_r () const**
  
  Returns whether a widget and all its parents are visible.

• int **w () const**
  
  Gets the widget width.

• Fl_When **when () const**
  
  Returns the conditions under which the callback is called.

• void **when (uchar i)**
  
  Sets the flags used to decide when a callback is called.

• Fl_Window ∗**window () const**
  
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int **x () const**
  
  Gets the widget position in its window.

• int **y () const**
  
  Gets the widget position in its window.

• virtual ∼**Fl_Widget ()**
  
  Destroys the widget.

**Additional Inherited Members**

**Public Types inherited from Fl_Browser**

• enum Fl_Line_Position { TOP, BOTTOM, MIDDLE }
  
  For internal use only?

**Public Types inherited from Fl_Browser_**

• enum {
  
  HORIZONTAL = 1, VERTICAL = 2, BOTH = 3, ALWAYS_ON = 4,
  HORIZONTAL_ALWAYS = 5, VERTICAL_ALWAYS = 6, BOTH_ALWAYS = 7 }

  Values for has_scrollbar().

**Static Public Member Functions inherited from Fl_Group**

• static Fl_Group ∗**current ()**
  
  Returns the currently active group.

• static void **current (Fl_Group ∗g)**
  
  Sets the current group.
Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Public Attributes inherited from Fl_Browser_

- Fl_Scrollbar hscrollbar
  Horizontal scrollbar.

- Fl_Scrollbar scrollbar
  Vertical scrollbar.

Protected Types inherited from Fl_Widget

```c
enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPYED_TOOLtip = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCErNTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
```
flags possible values enumeration.

Protected Member Functions inherited from Fl_Browser

- FL_BLINE * _remove (int line)
  Removes the item at the specified line.

- FL_BLINE * find_line (int line) const
  Returns the item for specified line.

- int full_height () const FL_OVERRIDE
  The height of the entire list of all visible() items in pixels.

- int incr_height () const FL_OVERRIDE
  The default 'average' item height (including inter-item spacing) in pixels.

- void insert (int line, FL_BLINE *item)
  Insert specified item above line.

- void * item_at (int line) const FL_OVERRIDE
  Return the item at specified line.

- void item_draw (void *item, int X, int Y , int W, int H) const FL_OVERRIDE
  Draws item at the position specified by XYWH.

- void * item_first () const FL_OVERRIDE
  Returns the very first item in the list.

- int item_height (void *item) const FL_OVERRIDE
  Returns height of item in pixels.

- void * item_last () const FL_OVERRIDE
Returns the very last item in the list.

- void item_next (void *item) const FL_OVERRIDEx Returns the next item after item.
- void item_prev (void *item) const FL_OVERRIDEx Returns the previous item before item.
- void item_select (void *item, int val) FL_OVERRIDEx Change the selection state of item to the value val.
- int item_selected (void *item) const FL_OVERRIDEx See if item is selected.
- void item_swap (void *a, void *b) FL_OVERRIDEx Swap the items a and b.
- const char * item_text (void *item) const FL_OVERRIDEx Returns the label text for item.
- int item_width (void *item) const FL_OVERRIDEx Returns width of item in pixels.
- int lineno (void *item) const FL_OVERRIDEx Returns line number corresponding to item, or zero if not found.
- void swap (FL_BLINE *a, FL_BLINE *b) FL_OVERRIDEx Swap the two items a and b.

Protected Member Functions inherited from Fl_Browser_

- void bbox (int &X, int &Y, int &W, int &H) const FL_OVERRIDEx Returns the bounding box for the interior of the list's display window, inside the scrollbars.
- void deleting (void *item) FL_OVERRIDEx This method should be used when item is being deleted from the list.
- int displayed (void *item) const FL_OVERRIDEx Returns non-zero if item has been scrolled to a position where it is being displayed.
- void draw () FL_OVERRIDEx Draws the list within the normal widget bounding box.
- void * find_item (int ypos) FL_OVERRIDEx This method returns the item under mouse y position ypos.
- Fl_Browser_ (int X, int Y, int W, int H, const char *L=0) FL_OVERRIDEx The constructor makes an empty browser.
- virtual int full_width () const FL_OVERRIDEx This method may be provided by the subclass to indicate the full width of the item list, in pixels.
- void inserting (void *a, void *b) FL_OVERRIDEx This method should be used when an item is in the process of being inserted into the list.
- virtual int item_quick_height (void *item) const FL_OVERRIDEx This method may be provided by the subclass to return the height of the item, in pixels.
- int leftedge () const FL_OVERRIDEx This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.
- void new_list () FL_OVERRIDEx This method should be called when the list data is completely replaced or cleared.
- void redraw_line (void *item) FL_OVERRIDEx This method should be called when the contents of item has changed, but not its height.
- void redraw_lines () FL_OVERRIDEx This method will cause the entire list to be redrawn.
- void replacing (void *a, void *b) FL_OVERRIDEx This method should be used when item a is being replaced by item b.
- void * selection () const FL_OVERRIDEx
Returns the item currently selected, or NULL if there is no selection.

- **void swapping (void ∗a, void ∗b)**
  This method should be used when two items a and b are being swapped.

- **void ∗top () const**
  Returns the item that appears at the top of the list.

**Protected Member Functions inherited from Fl_Group**

- **Fl_Rect ∗bounds ()**
  Returns the internal array of widget sizes and positions.

- **void draw () FL_OVERRIDE**
  Draws the widget.

- **void draw_child (Fl_Widget &widget) const**
  Forces a child to redraw.

- **void draw_children ()**
  Draws all children of the group.

- **void draw_outside_label (const Fl_Widget &widget) const**
  Parents normally call this to draw outside labels of child widgets.

- **virtual int on_insert (Fl_Widget ∗, int)**
  Allow derived groups to act when a widget is added as a child.

- **virtual int on_move (int, int)**
  Allow derived groups to act when a widget is moved within the group.

- **virtual void on_remove (int)**
  Allow derived groups to act when a child widget is removed from the group.

- **int ∗sizes ()**
  Returns the internal array of widget sizes and positions.

- **void update_child (Fl_Widget &widget) const**
  Draws a child only if it needs it.

**Protected Member Functions inherited from Fl_Widget**

- **void clear_flag (unsigned int c)**
  Clears a flag in the flags mask.

- **void draw_backdrop () const**
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- **void draw_box () const**
  Draws the widget box according its box style.

- **void draw_box (Fl_Boxtype t, Fl_Color c) const**
  Draws a box of type t, of color c at the widget's position and size.

- **void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const**
  Draws a box of type t, of color c at the position X,Y and size W,H.

- **void draw_focus () const**
  Draws a focus rectangle around the widget.

- **void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const**
  Draws a focus rectangle around the widget.

- **void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const**
  Draws a focus box for the widget at the given position and size.

- **void draw_label () const**
  Draws the widget's label at the defined label position.

- **void draw_label (int, int, int, int) const**
  Draws the label in an arbitrary bounding box.

- **Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)**
Creates a widget at the given position and size.

- unsigned int flags () const
  
  Gets the widget flags mask.

- void h (int v)
  
  Internal use only.

- void set_flag (unsigned int c)
  
  Sets a flag in the flags mask.

- void w (int v)
  
  Internal use only.

- void x (int v)
  
  Internal use only.

- void y (int v)
  
  Internal use only.

12.65.1 Detailed Description

The Fl_Hold_Browser is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space.

![Figure 12.24 Fl_Hold_Browser](image)

As long as the mouse button is held down the item pointed to by it is highlighted, and this highlighting remains on when the mouse button is released. Normally the callback is done when the user releases the mouse, but you can change this with when().

See Fl_Browser for methods to add and remove lines from the browser.

12.65.2 Constructor & Destructor Documentation

12.65.2.1 Fl_Hold_Browser()

```cpp
Fl_Hold_Browser::Fl_Hold_Browser ( int X,
                                   int Y,
                                   int W,
                                   int H,
                                   const char * L = 0 )
```

Creates a new Fl_Hold_Browser widget using the given position, size, and label string.

The default boxtype is FL_DOWN_BOX. The constructor specializes Fl_Browser() by setting the type to FL_HOLD_BROWSER. The destructor destroys the widget and frees all memory that has been allocated.

The documentation for this class was generated from the following files:

- Fl_Hold_Browser.H
- Fl_Browser.cxx

12.66 Fl_Hor_Fill_Slider Class Reference

Inheritance diagram for Fl_Hor_Fill_Slider:
Public Member Functions

- **Fl_Hor_Fill_Slider** (int X, int Y, int W, int H, const char *L=0)

Public Member Functions inherited from Fl_Slider

- void **bounds** (double a, double b)
  - Sets the minimum (a) and maximum (b) values for the valuator widget.
- **Fl_Slider** (int X, int Y, int W, int H, const char *L=0)
  - Creates a new Fl_Slider widget using the given position, size, and label string.
- **Fl_Slider** (uchar t, int X, int Y, int W, int H, const char *L)
  - Creates a new Fl_Slider widget using the given type, position, size, and label string.
- int **handle** (int) FL_OVERRIDE
  - Handles the specified event.
- int **scrollvalue** (int pos, int size, int first, int total)
  - Sets the size and position of the sliding knob in the box.
- **Fl_Boxtype** **slider** () const
  - Gets the slider box type.
- void **slider** (Fl_Boxtype c)
  - Sets the slider box type.
- float **slider_size** () const
  - Get the dimensions of the moving piece of slider.
- void **slider_size** (double v)
  - Set the dimensions of the moving piece of slider.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)
  - Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)
  - Clamps the passed value to the valuator range.
- virtual int **format** (char *)
  - Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- double **increment** (double, int)
  - Adds n times the step value to the passed value.
- double **maximum** () const
  - Gets the maximum value for the valuator.
- void **maximum** (double a)
  - Sets the maximum value for the valuator.
- double **minimum** () const
  - Gets the minimum value for the valuator.
- void **minimum** (double a)
Sets the minimum value for the valuator.

- void precision (int digits)
  Sets the step value to $1.0 / 10^{\text{digits}}$.

- void range (double a, double b)
  Sets the minimum and maximum values for the valuator.

- double round (double)
  Round the passed value to the nearest step increment.

- double step () const
  Gets or sets the step value.

- void step (double a, int b)
  See double Fl_Valuator::step() const

- void step (double s)
  See double Fl_Valuator::step() const.

- void step (int a)
  See double Fl_Valuator::step() const

- double value () const
  Gets the floating point(double) value.

- int value (double)
  Sets the current value.

### Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.

- unsigned int active () const
  Returns whether the widget is active.

- int active_r () const
  Returns whether the widget and all of its parents are active.

- Fl_Align align () const
  Gets the label alignment.

- void align (Fl_Align alignment)
  Sets the label alignment.

- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.

- virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const * as_gl_window () const

- virtual Fl_Group * as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- virtual Fl_Group const * as_group () const

- virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const * as_window () const

- void bind_deimage (Fl_Image * img)
  Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage (int f)
Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- **void bind_image(Fl_Image *img)**
  - Sets the image to use as part of the widget label when in the active state.

- **void bind_image(int f)**
  - Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype box() const**
  - Gets the box type of the widget.

- **void box(Fl_Boxtype new_box)**
  - Sets the box type for the widget.

- **Fl_Callback *callback() const**
  - Gets the current callback function for the widget.

- **void callback(Fl_Callback *cb)**
  - Sets the current callback function for the widget.

- **void callback(Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)**
  - Sets the current callback function and managed user data for the widget.

- **void callback(Fl_Callback *cb, void *p)**
  - Sets the current callback function and data for the widget.

- **void callback(Fl_Callback0 *cb)**
  - Sets the current callback function for the widget.

- **void callback(Fl_Callback1 *cb, long p=0)**
  - Sets the current callback function for the widget.

- **unsigned int changed() const**
  - Checks if the widget value changed since the last callback.

- **void clear_active()**
  - Marks the widget as inactive without sending events or changing focus.

- **void clear_changed()**
  - Marks the value of the widget as unchanged.

- **void clear_damage(uchar c=0)**
  - Clears or sets the damage flags.

- **void clear_output()**
  - Sets a widget to accept input.

- **void clear_visible()**
  - Hides the widget.

- **void clear_visible_focus()**
  - Disables keyboard focus navigation with this widget.

- **Fl_Color color() const**
  - Gets the background color of the widget.

- **void color(Fl_Color bg)**
  - Sets the background color of the widget.

- **void color(Fl_Color bg, Fl_Color sel)**
  - Sets the background and selection color of the widget.

- **Fl_Color color2() const**
  - For back compatibility only.

- **void color2(unsigned a)**
  - For back compatibility only.

- **int contains(const Fl_Widget *w) const**
  - Checks if w is a child of this widget.

- **void copy_label(const char *new_label)**
  - Sets the current label.

- **void copy_tooltip(const char *text)**
  - Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image * deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image * deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
Shortcut to set the label text and type in one call.

- `Fl_Color labelcolor () const`
  - Gets the label color.

- `void labelcolor (Fl_Color c)`
  - Sets the label color.

- `Fl_Font labelfont () const`
  - Gets the font to use.

- `void labelfont (Fl_Font f)`
  - Sets the font to use.

- `Fl_Fontsize labelsizes () const`
  - Gets the font size in pixels.

- `void labelsizes (Fl_Fontsize pix)`
  - Sets the font size in pixels.

- `Fl_Labeltype labelfontype () const`
  - Gets the label type.

- `void labelfontype (Fl_Labeltype a)`
  - Sets the label type.

- `void measure_label (int &ww, int &hh) const`
  - Sets width ww and height hh accordingly with the label size.

- `bool needs_keyboard () const`
  - Returns whether this widget needs a keyboard.

- `void needs_keyboard (bool needs)`
  - Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  - Returns if a widget is used for output only.

- `Fl_Group *parent () const`
  - Returns a pointer to the parent widget.

- `void parent (Fl_Group *p)`
  - Internal use only - "for hacks only".

- `void position (int X, int Y)`
  - Repositions the window or widget.

- `void redraw ()`
  - Schedules the drawing of the widget.

- `void redraw_label ()`
  - Schedules the drawing of the label.

- `virtual void resize (int x, int y, int w, int h)`
  - Changes the size or position of the widget.

- `Fl_Color selection_color () const`
  - Gets the selection color.

- `void selection_color (Fl_Color a)`
  - Sets the selection color.

- `void set_active ()`
  - Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  - Marks the value of the widget as changed.

- `void set_output ()`
  - Sets a widget to output only.

- `void set_visible ()`
  - Makes the widget visible.

- `void set_visible_focus ()`
  - Enables keyboard focus focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int testShortcut ()
  Returns true if the widget’s label contains the entered ‘&x’ shortcut.
• const char ∗ tooltip () const
  Gets the current tooltip text.
• void tooltip (const char ∗text)
  Sets the current tooltip text.
• Fl_Window ∗ top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void ∗ user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void ∗v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window ∗ window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
Gets the widget position in its window.

- `int y () const`

Gets the widget position in its window.

- `virtual ~Fl_Widget ()`
  Destroys the widget.

**Additional Inherited Members**

**Static Public Member Functions inherited from Fl_Widget**

- `static void default_callback (Fl_Widget *,void *)`
  The default callback for all widgets that don't set a callback.

- `static unsigned int label_shortcut (const char *)`
  Returns the Unicode value of the '&x' shortcut in a given text.

- `static int test_shortcut (const char *, const bool require_alt=false)`
  Returns true if the given text \( t \) contains the entered '&x' shortcut.

**Protected Types inherited from Fl_Widget**

- `enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<25 , USERFLAG2 = 1<<26 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

**Protected Member Functions inherited from Fl_Slider**

- `void draw () FL_OVERRIDE`
  Draws the widget.

- `void draw (int, int, int, int)`
- `int handle (int, int, int, int)`

**Protected Member Functions inherited from Fl_Valuator**

- `Fl_Valuator (int X, int Y , int W, int H, const char *)`
  Creates a new Fl_Valuator widget using the given position, size, and label string.

- `void handle_drag (double newvalue)`
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.

- `void handle_push ()`
  Stores the current value in the previous value.

- `void handle_release ()`
  Called after an FL_WHEN_RELEASE event is received and before the callback.

- `int horizontal () const`
  Tells if the valuator is an FL_HORIZONTAL one.

- `double previous_value () const`
  Gets the previous floating point value before an event changed it.
• **void** **set_value**(double v)
  
  Sets the current floating point value.

• **double** **softclamp**(double)
  
  Clamps the value, but accepts v if the previous value is not already out of range.

• **virtual void** **value_damage**()
  
  Asks for partial redraw.

**Protected Member Functions inherited from Fl_Widget**

• **void** **clear_flag**(unsigned int c)
  
  Clears a flag in the flags mask.

• **void** **draw_backdrop**(const)
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• **void** **draw_box**(Fl_Boxtype t, Fl_Color c) const
  
  Draws a box of type t, of color c at the widget’s position and size.

• **void** **draw_box**(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type t, of color c at the position X,Y and size W,H.

• **void** **draw_focus**(const)
  
  Draws a focus rectangle around the widget.

• **void** **draw_focus**(Fl_Boxtype t, int X, int Y, int W, int H) const
  
  Draws a focus rectangle around the widget.

• **void** **draw_focus**(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.

• **void** **draw_label**(const)
  
  Draws the widget’s label at the defined label position.

• **void** **draw_label**(int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.

• **Fl_Widget**(int x, int y, int w, int h, const char *label=0L)
  
  Creates a widget at the given position and size.

• **unsigned int** **flags**(const)
  
  Gets the widget flags mask.

• **void** **h**(int v)
  
  Internal use only.

• **void** **set_flag**(unsigned int c)
  
  Sets a flag in the flags mask.

• **void** **w**(int v)
  
  Internal use only.

• **void** **x**(int v)
  
  Internal use only.

• **void** **y**(int v)
  
  Internal use only.

The documentation for this class was generated from the following files:

- Fl_Hor_Fill_Slider.H
- Fl_Slider.cxx
**12.67  Fl_Hor_Nice_Slider Class Reference**

Single thumb tab slider.

```
#include <Fl_Hor_Nice_Slider.H>
```

Inheritance diagram for Fl_Hor_Nice_Slider:

```
Fl_Widget
    ▼
     ▼
Fl_Valuator
    ▼
Fl_Slider
    ▼
Fl_Hor_Nice_Slider
```

**Public Member Functions**

- `Fl_Hor_Nice_Slider(int X, int Y, int W, int H, const char *L=0)`

**Public Member Functions inherited from Fl_Slider**

- `void bounds(double a, double b)`
  Sets the minimum (a) and maximum (b) values for the valuator widget.
- `Fl_Slider(int X, int Y, int W, int H, const char *L=0)`
  Creates a new Fl_Slider widget using the given position, size, and label string.
- `Fl_Slider(uchar t, int X, int Y, int W, int H, const char *L)`
  Creates a new Fl_Slider widget using the given type, position, size, and label string.
- `int handle(int)` FL_OVERRIDE
  Handles the specified event.
- `int scrollvalue(int pos, int size, int first, int total)`
  Sets the size and position of the sliding knob in the box.
- `Fl_Boxtype slider() const`
  Gets the slider box type.
- `void slider(Fl_Boxtype c)`
  Sets the slider box type.
- `float slider_size() const`
  Gets the dimensions of the moving piece of slider.
- `void slider_size(double v)`
  Set the dimensions of the moving piece of slider.

**Public Member Functions inherited from Fl_Valuator**

- `void bounds(double a, double b)`
  Sets the minimum (a) and maximum (b) values for the valuator widget.
- `double clamp(double)`
  Clamps the passed value to the valuator range.
- `virtual int format(char *)`
  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- `double increment(double, int)`
  Adds n times the step value to the passed value.
- `double maximum() const`
  Gets the maximum value for the valuator.
• void **maximum** (double a)
  
  *Sets the maximum value for the valuator.*

• double **minimum** () const
  
  *Gets the minimum value for the valuator.*

• void **minimum** (double a)
  
  *Sets the minimum value for the valuator.*

• void **precision** (int digits)
  
  *Sets the step value to 1.0 / 10^digits.*

• void **range** (double a, double b)
  
  *Sets the minimum and maximum values for the valuator.*

• double **round** (double)
  
  *Round the passed value to the nearest step increment.*

• double **step** () const
  
  *Gets or sets the step value.*

• void **step** (double a, int b)
  
  See double **Fl_Valuator::step()** const

• void **step** (double s)
  
  See double **Fl_Valuator::step()** const.

• void **step** (int a)
  
  See double **Fl_Valuator::step()** const

• double **value** () const
  
  *Gets the floating point(double) value.*

• int **value** (double)
  
  *Sets the current value.*

### Public Member Functions inherited from Fl_Widget

• void **_clearFullscreen** ()

• void **_setFullscreen** ()

• void **activate** ()
  
  *Activates the widget.*

• unsigned int **active** () const
  
  *Returns whether the widget is active.*

• int **active_r** () const
  
  *Returns whether the widget and all of its parents are active.*

• **Fl_Align align** () const
  
  *Gets the label alignment.*

• void **align** (Fl_Align alignment)
  
  *Sets the label alignment.*

• long **argument** () const
  
  *Gets the current user data (long) argument that is passed to the callback function.*

• void **argument** (long v)
  
  *Sets the current user data (long) argument that is passed to the callback function.*

• virtual class Fl_Gl_Window **as_gl_window** ()
  
  *Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.*

• virtual class Fl_Gl_Window const **as_gl_window** () const

• virtual Fl_Group **as_group** ()
  
  *Returns an Fl_Group pointer if this widget is an Fl_Group.*

• virtual Fl_Group const **as_group** () const

• virtual Fl_Window **as_window** () const

Generated by Doxygen
Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const & as_window() const
  - Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage(Fl_Image * img)
  - Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image(Fl_Image * img)
  - Sets the image to use as part of the widget label when in the active state.
- void bind_image(int f)
  - Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box() const
  - Gets the box type of the widget.
- void box(Fl_Boxtype new_box)
  - Sets the box type for the widget.
- Fl_Callback_p callback() const
  - Gets the current callback function for the widget.
- void callback(Fl_Callback * cb)
  - Sets the current callback function for the widget.
- void callback(Fl_Callback * cb, Fl_Callback_User_Data * p, bool auto_free)
  - Sets the current callback function and managed user data for the widget.
- void callback(Fl_Callback * cb, void * p)
  - Sets the current callback function and data for the widget.
- void callback(Fl_Callback0 * cb)
  - Sets the current callback function for the widget.
- void callback(Fl_Callback1 * cb, long p=0)
  - Sets the current callback function for the widget.
- unsigned int changed() const
  - Checks if the widget value changed since the last callback.
- void clear_active()
  - Marks the widget as inactive without sending events or changing focus.
- void clear_changed()
  - Marks the value of the widget as unchanged.
- void clear_damage(uchar c=0)
  - Clears or sets the damage flags.
- void clear_output()
  - Sets a widget to accept input.
- void clear_visible()
  - Hides the widget.
- void clear_visible_focus()
  - Disables keyboard focus navigation with this widget.
- Fl_Color color() const
  - Gets the background color of the widget.
- void color(Fl_Color bg)
  - Sets the background color of the widget.
- void color(Fl_Color bg, Fl_Color sel)
  - Sets the background and selection color of the widget.
- Fl_Color color2() const
  - For back compatibility only.
- void color2(unsigned a)
  - For back compatibility only.
- int contains(const Fl_Widget * w) const
Checks if w is a child of this widget.

- **void copy_label (const char ∗new_label)**
  Sets the current label.

- **void copy_tooltip (const char ∗text)**
  Sets the current tooltip text.

- **uchar damage () const**
  Returns non-zero if draw() needs to be called.

- **void damage (uchar c)**
  Sets the damage bits for the widget.

- **void damage (uchar c, int x, int y, int w, int h)**
  Sets the damage bits for an area inside the widget.

- **int damage_resize (int, int, int, int)**
  Internal use only.

- **void deactivate ()**
  Deactivates the widget.

- **Fl_Image ∗deimage ()**
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image ∗deimage () const**
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (Fl_Image &img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (Fl_Image ∗img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const**
  Returns whether the inactive image is managed by the widget.

- **void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with default arguments.

- **void do_callback (Fl_Widget ∗widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void do_callback (Fl_Widget ∗widget, void ∗arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void draw_label (int, int, int, int, Fl_Align) const**
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h () const**
  Gets the widget height.

- **virtual void hide ()**
  Makes a widget invisible.

- **Fl_Image ∗image ()**
  Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image ∗image () const**
  Gets the image that is used as part of the widget label when in the active state.

- **void image (Fl_Image &img)**
  Sets the image to use as part of the widget label when in the active state.

- **void image (Fl_Image ∗img)**
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const**
  Returns whether the image is managed by the widget.

- **int inside (const Fl_Widget ∗wgt) const**
  Checks if this widget is a child of wgt.

- **int is_label_copied () const**
  Returns whether the current label was assigned with copy_label().

Generated by Doxygen
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - “for hacks only”.
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
Sets a widget to output only.

- **void set_visible ()**
  Makes the widget visible.

- **void set_visible_focus ()**
  Enables keyboard focus navigation with this widget.

- **int shortcut_label () const**
  Returns whether the widget's label uses '&' to indicate shortcuts.

- **void shortcut_label (int value)**
  Sets whether the widget's label uses '&' to indicate shortcuts.

- **virtual void show ()**
  Makes a widget visible.

- **void size (int W, int H)**
  Changes the size of the widget.

- **int take_focus ()**
  Gives the widget the keyboard focus.

- **unsigned int takesevents () const**
  Returns if the widget is able to take events.

- **int test_shortcut ()**
  Returns true if the widget's label contains the entered 'x' shortcut.

- **const char * tooltip () const**
  Gets the current tooltip text.

- **void tooltip (const char *text)**
  Sets the current tooltip text.

- **Fl_Window * top_window () const**
  Returns a pointer to the top-level window for the widget.

- **Fl_Window * top_window_offset (int &xoff, int &yoff) const**
  Finds the x/y offset of the current widget relative to the top-level window.

- **uchar type () const**
  Gets the widget type.

- **void type (uchar t)**
  Sets the widget type.

- **int use_accents_menu ()**
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- **void * user_data () const**
  Gets the user data for this widget.

- **void user_data (Fl_Callback_User_Data *v, bool auto_free)**
  Sets the user data for this widget.

- **void user_data (void *v)**
  Sets the user data for this widget.

- **unsigned int visible () const**
  Returns whether a widget is visible.

- **unsigned int visible_focus () const**
  Checks whether this widget has a visible focus.

- **void visible_focus (int v)**
  Modifies keyboard focus navigation.

- **int visible_r () const**
  Returns whether a widget and all its parents are visible.

- **int w () const**
  Gets the widget width.

- **Fl_When when () const**
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget * widget, void *data)
  The default callback for all widgets that don’t set a callback.
• static unsigned int label_shortcut (const char * t)
  Returns the Unicode value of the ’&x’ shortcut in a given text.
• static int test_shortcut (const char * t, const bool require_alt=false)
  Returns true if the given text t contains the entered ’&x’ shortcut.

Protected Types inherited from Fl_Widget

• enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22, AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
flags possible values enumeration.

Protected Member Functions inherited from Fl_Slider

• void draw () FL_OVERRIDE
  Draws the widget.
• void draw (int, int, int, int)
• int handle (int, int, int, int)

Protected Member Functions inherited from Fl_Valuator

• Fl_Valuator (int X, int Y, int W, int H, const char *L)
  Creates a new Fl_Valuator widget using the given position, size, and label string.
• void handle_drag (double newvalue)
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
• void handle_push ()
  Stores the current value in the previous value.
• void handle_release ()

Generated by Doxygen
Called after an FL_WHEN_RELEASE event is received and before the callback.

- **int horizontal() const**
  
  Tells if the valuator is an FL_HORIZONTAL one.

- **double previous_value() const**
  
  Gets the previous floating point value before an event changed it.

- **void set_value(double v)**
  
  Sets the current floating point value.

- **double softclamp(double)**
  
  Clamps the value, but accepts v if the previous value is not already out of range.

- **virtual void value_damage()**
  
  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- **void clear_flag(unsigned int c)**
  
  Clears a flag in the flags mask.

- **void draw_backdrop() const**
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- **void draw_box() const**
  
  Draws the widget box according its box style.

- **void draw_box(Fl_Boxtype t, Fl_Color c) const**
  
  Draws a box of type t, of color c at the widget's position and size.

- **void draw_box(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const**
  
  Draws a box of type t, of color c at the position X,Y and size W,H.

- **void draw_focus() const**
  
  Draws a focus rectangle around the widget.

- **void draw_focus(Fl_Boxtype t, int X, int Y, int W, int H) const**
  
  Draws a focus rectangle around the widget.

- **void draw_focus(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const**
  
  Draws a focus box for the widget at the given position and size.

- **void draw_label() const**
  
  Draws the widget's label at the defined label position.

- **void draw_label(int, int, int, int) const**
  
  Draws the label in an arbitrary bounding box.

- **void Fl_Widget(int x, int y, int w, int h, const char *label=0L)**
  
  Creates a widget at the given position and size.

- **unsigned int flags() const**
  
  Gets the widget flags mask.

- **void h(int v)**
  
  Internal use only.

- **void set_flag(unsigned int c)**
  
  Sets a flag in the flags mask.

- **void w(int v)**
  
  Internal use only.

- **void x(int v)**
  
  Internal use only.

- **void y(int v)**
  
  Internal use only.
12.67.1 Detailed Description

Single thumb tab slider.

![Figure 12.25 Fl_Hor_Nice_Slider with various Fl::scheme() values](image)

The documentation for this class was generated from the following files:

- Fl_Hor_Nice_Slider.H
- Fl_Slider.cxx

12.68 Fl_Hor_Slider Class Reference

Horizontal Slider class.

```cpp
#include <Fl_Hor_Slider.H>
```

Inheritance diagram for Fl_Hor_Slider:

```
Fl_Widget
    ↓
Fl_Valuator
    ↓
Fl_Slider
    ↓
Fl_Hor_Slider
```

Public Member Functions

- **Fl_Hor_Slider** (int X, int Y, int W, int H, const char *l=0)
  
  Creates a new Fl_Hor_Slider widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Slider

- void bounds (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.

- Fl_Slider (int X, int Y, int W, int H, const char *L=0)

  Creates a new Fl_Slider widget using the given position, size, and label string.

- Fl_Slider (uchar t, int X, int Y, int W, int H, const char *L)

  Creates a new Fl_Slider widget using the given type, position, size, and label string.

- int handle (int) FL_OVERRIDE

  Handles the specified event.

- int scrollvalue (int pos, int size, int first, int total)

  Sets the size and position of the sliding knob in the box.

- **Fl_Boxtype slider ()** const

  Gets the slider box type.

- void slider (Fl_Boxtype c)
Sets the slider box type.

- float `slider_size()` const
  
  Get the dimensions of the moving piece of slider.

- void `slider_size(double v)`
  
  Set the dimensions of the moving piece of slider.

### Public Member Functions inherited from `Fl_Valuator`

- void `bounds(double a, double b)`
  
  Sets the minimum (a) and maximum (b) values for the valuator widget.

- double `clamp(double)`
  
  Clamps the passed value to the valuator range.

- virtual int `format(char*)`
  
  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.

- double `increment(double, int)`
  
  Adds n times the step value to the passed value.

- double `maximum()` const
  
  Gets the maximum value for the valuator.

- void `maximum(double a)`
  
  Sets the maximum value for the valuator.

- double `minimum()` const
  
  Gets the minimum value for the valuator.

- void `minimum(double a)`
  
  Sets the minimum value for the valuator.

- void `precision(int digits)`
  
  Sets the step value to $1.0 / 10^{\text{digits}}$.

- void `range(double a, double b)`
  
  Sets the minimum and maximum values for the valuator.

- double `round(double)`
  
  Round the passed value to the nearest step increment.

- double `step()` const
  
  Gets or sets the step value.

- void `step(double a, int b)`
  
  See double `Fl_Valuator::step() const`

- void `step(double s)`
  
  See double `Fl_Valuator::step() const`.

- void `step(int a)`
  
  See double `Fl_Valuator::step() const`

- double `value()` const
  
  Gets the floating point(double) value.

- int `value(double)`
  
  Sets the current value.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ as_gl_window () const
- virtual Fl_Group ∗ as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ as_group () const
- virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ as_window () const
- void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
  Returns the current user data (long) argument that is passed to the callback function.
Checks if the widget value changed since the last callback.
• void clear_active ()
   Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
   Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
   Clears or sets the damage flags.
• void clear_output ()
   Sets a widget to accept input.
• void clear_visible ()
   Hides the widget.
• void clear_visible_focus ()
   Disables keyboard focus navigation with this widget.
• Fl_Color color () const
   Gets the background color of the widget.
• void color (Fl_Color bg)
   Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
   Sets the background and selection color of the widget.
• Fl_Color color2 () const
   For back compatibility only.
• void color2 (unsigned a)
   For back compatibility only.
• int contains (const Fl_Widget ∗w) const
   Checks if w is a child of this widget.
• void copy_label (const char ∗new_label)
   Sets the current label.
• void copy_tooltip (const char ∗text)
   Sets the current tooltip text.
• uchar damage () const
   Returns non-zero if draw() needs to be called.
• void damage (uchar c)
   Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
   Sets the damage bits for an area inside the widget.
• int damage_size (int, int, int)
   Internal use only.
• void deactivate ()
   Deactivates the widget.
• Fl_Image ∗deimage ()
   Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image ∗deimage () const
   Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
   Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image ∗img)
   Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
   Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
   Calls the widget callback function with default arguments.
void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
Calls the widget callback function with arbitrary arguments.

void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
Calls the widget callback function with arbitrary arguments.

void draw_label (int, int, int, int, Fl_Align) const
Draws the label in an arbitrary bounding box with an arbitrary alignment.

int h () const
Gets the widget height.

virtual void hide ()
Makes a widget invisible.

Fl_Image *image ()
Gets the image that is used as part of the widget label when in the active state.

const Fl_Image *image () const
Gets the image that is used as part of the widget label when in the active state.

void image (Fl_Image &img)
Sets the image to use as part of the widget label when in the active state.

void image (Fl_Image *img)
Sets the image to use as part of the widget label when in the active state.

int image_bound () const
Returns whether the image is managed by the widget.

int inside (const Fl_Widget *wgt) const
Checks if this widget is a child of wgt.

int is_label_copied () const
Returns whether the current label was assigned with copy_label().

const char *label () const
Gets the current label text.

void label (const char *text)
Sets the current label pointer.

void label (Fl_Labeltype a, const char *b)
Shortcut to set the label text and type in one call.

Fl_Color labelcolor () const
Gets the label color.

void labelcolor (Fl_Color c)
Sets the label color.

Fl_Font labelfont () const
Gets the font to use.

void labelfont (Fl_Font f)
Sets the font to use.

Fl_Fontsize labelsize () const
Gets the font size in pixels.

void labelsize (Fl_Fontsize pix)
Sets the font size in pixels.

Fl_Labeltype labeltype () const
Gets the label type.

void labeltype (Fl_Labeltype a)
Sets the label type.

void measure_label (int &ww, int &hh) const
Sets width ww and height hh accordingly with the label size.

bool needs_keyboard () const
Returns whether this widget needs a keyboard.

void needs_keyboard (bool needs)
Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  
  Returns if a widget is used for output only.

- `Fl_Group * parent () const`
  
  Returns a pointer to the parent widget.

- `void parent (Fl_Group *p)`
  
  Internal use only - "for hacks only".

- `void position (int X, int Y)`
  
  Repositions the window or widget.

- `void redraw ()`
  
  Schedules the drawing of the widget.

- `void redraw_label ()`
  
  Schedules the drawing of the label.

- `virtual void resize (int x, int y, int w, int h)`
  
  Changes the size or position of the widget.

- `Fl_Color selection_color () const`
  
  Gets the selection color.

- `void selection_color (Fl_Color a)`
  
  Sets the selection color.

- `void set_active ()`
  
  Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  
  Marks the value of the widget as changed.

- `void set_output ()`
  
  Sets a widget to output only.

- `void set_visible ()`
  
  Makes the widget visible.

- `void set_visible_focus ()`
  
  Enables keyboard focus navigation with this widget.

- `int shortcut_label () const`
  
  Returns whether the widget's label uses '&' to indicate shortcuts.

- `void shortcut_label (int value)`
  
  Sets whether the widget's label uses '&' to indicate shortcuts.

- `virtual void show ()`
  
  Makes a widget visible.

- `void size (int W, int H)`
  
  Changes the size of the widget.

- `int take_focus ()`
  
  Gives the widget the keyboard focus.

- `unsigned int takesevents () const`
  
  Returns if the widget is able to take events.

- `int test_shortcut ()`
  
  Returns true if the widget's label contains the entered '&x' shortcut.

- `const char * tooltip () const`
  
  Gets the current tooltip text.

- `void tooltip (const char *text)`
  
  Sets the current tooltip text.

- `Fl_Window * top_window () const`
  
  Returns a pointer to the top-level window for the widget.

- `Fl_Window * top_window_offset (int &xoff, int &yoff) const`
  
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• FL_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• FL_Window *window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don’t set a callback.
• static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the ’&x’ shortcut in a given text.
• static int test_shortcut (const char *t, const bool require_alt=false)
  Returns true if the given text t contains the entered ’&x’ shortcut.
Protected Types inherited from Fl_Widget

```c
enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLKIT = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
```

flags possible values enumeration.

Protected Member Functions inherited from Fl_Slider

- void draw () FL_OVERRIDE
  Draws the widget.
- void draw (int, int, int, int)
- int handle (int, int, int, int)

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y , int W, int H, const char ∗L)
  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void handle_drag (double newvalue)
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void handle_push ()
  Stores the current value in the previous value.
- void handle_release ()
  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int horizontal () const
  Tells if the valuator is an FL_HORIZONTAL one.
- double previous_value () const
  Gets the previous floating point value before an event changed it.
- void set_value (double v)
  Sets the current floating point value.
- double softclamp (double)
  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()
  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
Draws a box of type t, of color c at the widget's position and size.

- void **draw_box** ([Fl_Boxtype](#) t, int x, int y, int w, int h, [Fl_Color](#) c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- void **draw_focus** () const
  Draws a focus rectangle around the widget.

- void **draw_focus** ([Fl_Boxtype](#) t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

- void **draw_focus** ([Fl_Boxtype](#) t, int x, int y, int w, int h, [Fl_Color](#) bg) const
  Draws a focus box for the widget at the given position and size.

- void **draw_label** () const
  Draws the widget's label at the defined label position.

- void **draw_label** (int, int, int, int) const
  Draws the label in an arbitrary bounding box.

- **FL_Widget** (int x, int y, int w, int h, const char +label=0L)
  Creates a widget at the given position and size.

- unsigned int **flags** () const
  Gets the widget flags mask.

- void **h** (int v)
  *Internal use only.*

- void **set_flag** (unsigned int c)
  Sets a flag in the flags mask.

- void **w** (int v)
  *Internal use only.*

- void **x** (int v)
  *Internal use only.*

- void **y** (int v)
  *Internal use only.*

### 12.68.1 Detailed Description

Horizontal Slider class.

See also

- class [Fl_Slider](#)

The documentation for this class was generated from the following files:

- Fl_Hor_Slider.H
- Fl_Slider.cxx

#### 12.69 Fl_Hor_Value_Slider Class Reference

Inheritance diagram for Fl_Hor_Value_Slider:

```
  Fl_Widget
     ▼
  Fl_Valuator
     ▼
  Fl_Slider
     ▼
  Fl_Value_Slider
     ▼
Fl_Hor_Value_Slider
```

**Generated by Doxygen**
Public Member Functions

- **Fl_Hor_Value_Slider** (int X, int Y, int W, int H, const char *l=0)

Public Member Functions inherited from Fl_Value_Slider

- **Fl_Value_Slider** (int x, int y, int w, int h, const char *l=0)
  
  Creates a new Fl_Value_Slider widget using the given position, size, and label string.

- **int handle** (int) FL_OVERRIDE
  
  Handles the specified event.

- **Fl_Color textcolor** () const
  
  Gets the color of the text in the value box.

- void **textcolor** (Fl_Color s)
  
  Sets the color of the text in the value box.

- **Fl_Font textfont** () const
  
  Gets the typeface of the text in the value box.

- void **textfont** (Fl_Font s)
  
  Sets the typeface of the text in the value box.

- **Fl_Fontsize textsize** () const
  
  Gets the size of the text in the value box.

- void **textsize** (Fl_Fontsize s)
  
  Sets the size of the text in the value box.

- **int value_height** () const
  
  Gets the height of the value box in pixels (vertical mode only).

- void **value_height** (int s)
  
  Sets the height of the value box in pixels (vertical mode only).

- **int value_width** () const
  
  Gets the width of the value box in pixels (horizontal mode only).

- void **value_width** (int s)
  
  Sets the width of the value box in pixels (horizontal mode only).

Public Member Functions inherited from Fl_Slider

- void **bounds** (double a, double b)
  
  Sets the minimum (a) and maximum (b) values for the valuator widget.

- **Fl_Slider** (int X, int Y, int W, int H, const char *l=0)
  
  Creates a new Fl_Slider widget using the given position, size, and label string.

- **Fl_Slider** (uchar t, int X, int Y, int W, int H, const char *l)
  
  Creates a new Fl_Slider widget using the given type, position, size, and label string.

- **int scrollvalue** (int pos, int size, int first, int total)
  
  Sets the size and position of the sliding knob in the box.

- **Fl_Boxtype slider** () const
  
  Gets the slider box type.

- void **slider** (Fl_Boxtype c)
  
  Sets the slider box type.

- float **slider_size** () const
  
  Get the dimensions of the moving piece of slider.

- void **slider_size** (double v)
  
  Set the dimensions of the moving piece of slider.
Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)
  
  *Sets the minimum (a) and maximum (b) values for the valuator widget.*

- double **clamp** (double)
  
  *Clamps the passed value to the valuator range.*

- virtual int **format** (char ∗)
  
  *Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.*

- double **increment** (double, int)
  
  *Adds n times the step value to the passed value.*

- double **maximum** () const
  
  *Gets the maximum value for the valuator.*

- void **maximum** (double a)
  
  *Sets the maximum value for the valuator.*

- double **minimum** () const
  
  *Gets the minimum value for the valuator.*

- void **minimum** (double a)
  
  *Sets the minimum value for the valuator.*

- void **precision** (int digits)
  
  *Sets the step value to 1.0 / 10^digits.*

- void **range** (double a, double b)
  
  *Sets the minimum and maximum values for the valuator.*

- double **round** (double)
  
  *Round the passed value to the nearest step increment.*

- double **step** () const
  
  *Gets or sets the step value.*

- void **step** (double a, int b)
  
  *See double Fl_Valuator::step() const*

- void **step** (double s)
  
  *See double Fl_Valuator::step() const.*

- void **step** (int a)
  
  *See double Fl_Valuator::step() const*

- double **value** () const
  
  *Gets the floating point(double) value.*

- int **value** (double)
  
  *Sets the current value.*

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()

- void **_set_fullscreen** ()

- void **activate** ()
  
  *Activates the widget.*

- unsigned int **active** () const
  
  *Returns whether the widget is active.*

- int **active_r** () const
  
  *Returns whether the widget and all of its parents are active.*

- Fl_Align **align** () const
  
  *Gets the label alignment.*

- void **align** (Fl_Align alignment)

---

Generated by Doxygen
Sets the label alignment.

- **long argument () const**
  Gets the current user data (long) argument that is passed to the callback function.

- **void argument (long v)**
  Sets the current user data (long) argument that is passed to the callback function.

- **virtual class Fl_Gl_Window * as_gl_window ()**
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- **virtual class Fl_Gl_Window const * as_gl_window () const**
  virtual Fl_Group * as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- **virtual Fl_Group const * as_group () const**
  virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- **virtual Fl_Window const * as_window () const**

- **void bind_deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **void bind_deimage (int f)**
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- **void bind_image (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the active state.

- **void bind_image (int f)**
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype box () const**
  Gets the box type of the widget.

- **void box (Fl_Boxtype new_box)**
  Sets the box type for the widget.

- **Fl_Callback_p callback () const**
  Gets the current callback function for the widget.

- **void callback (Fl_Callback *cb)**
  Sets the current callback function for the widget.

- **void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)**
  Sets the current callback function and managed user data for the widget.

- **void callback (Fl_Callback *cb, void *p)**
  Sets the current callback function and data for the widget.

- **void callback (Fl_Callback0 *cb)**
  Sets the current callback function for the widget.

- **void callback (Fl_Callback1 *cb, long p=0)**
  Sets the current callback function for the widget.

- **unsigned int changed () const**
  Checks if the widget value changed since the last callback.

- **void clear_active ()**
  Marks the widget as inactive without sending events or changing focus.

- **void clear_changed ()**
  Marks the value of the widget as unchanged.

- **void clear_damage (uchar c=0)**
  Clears or sets the damage flags.

- **void clear_output ()**
  Sets a widget to accept input.

- **void clear_visible ()**
  Hides the widget.

- **void clear_visible_focus ()**
Disables keyboard focus navigation with this widget.

- `Fl_Color color () const`
  Gets the background color of the widget.

- `void color (Fl_Color bg)`
  Sets the background color of the widget.

- `void color (Fl_Color bg, Fl_Color sel)`
  Sets the background and selection color of the widget.

- `Fl_Color color2 () const`
  For back compatibility only.

- `void color2 (unsigned a)`
  For back compatibility only.

- `int contains (const Fl_Widget *w) const`
  Checks if w is a child of this widget.

- `void copy_label (const char *new_label)`
  Sets the current label.

- `void copy_tooltip (const char *text)`
  Sets the current tooltip text.

- `uchar damage () const`
  Returns non-zero if draw() needs to be called.

- `void damage (uchar c)`
  Sets the damage bits for the widget.

- `void damage (uchar c, int x, int y, int w, int h)`
  Sets the damage bits for an area inside the widget.

- `int damage_resize (int, int, int, int)`
  Internal use only.

- `void deactivate ()`
  Deactivates the widget.

- `Fl_Image *deimage ()`
  Gets the image that is used as part of the widget label when in the inactive state.

- `const Fl_Image *deimage () const`
  Gets the image that is used as part of the widget label when in the inactive state.

- `void deimage (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the inactive state.

- `void deimage (Fl_Image *img)`
  Sets the image to use as part of the widget label when in the inactive state.

- `int deimage_bound () const`
  Returns whether the inactive image is managed by the widget.

- `void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with default arguments.

- `void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.

- `void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.

- `void draw_label (int, int, int, int, Fl_Align) const`
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- `int h () const`
  Gets the widget height.

- `virtual void hide ()`
  Makes a widget invisible.

- `Fl_Image *image ()`
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image *image () const
  
  Gets the image that is used as part of the widget label when in the active state.

• void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the active state.

• void image (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the active state.

• int image_bound () const
  
  Returns whether the image is managed by the widget.

• int inside (const Fl_Widget *wgt) const
  
  Checks if this widget is a child of wgt.

• int is_label_copied () const
  
  Returns whether the current label was assigned with copy_label().

• const char * label () const
  
  Gets the current label text.

• void label (const char *text)
  
  Sets the current label pointer.

• void label (Fl_Labeltype a, const char *b)
  
  Shortcut to set the label text and type in one call.

• Fl_Color labelcolor () const
  
  Gets the label color.

• void labelcolor (Fl_Color c)
  
  Sets the label color.

• Fl_Font labelfont () const
  
  Gets the font to use.

• void labelfont (Fl_Font f)
  
  Sets the font to use.

• Fl_Fontsize labelsize () const
  
  Gets the font size in pixels.

• void labelsize (Fl_Fontsize pix)
  
  Sets the font size in pixels.

• Fl_Labeltype labeltype () const
  
  Gets the label type.

• void labeltype (Fl_Labeltype a)
  
  Sets the label type.

• void measure_label (int &ww, int &hh) const
  
  Sets width ww and height hh accordingly with the label size.

• bool needs_keyboard () const
  
  Returns whether this widget needs a keyboard.

• void needs_keyboard (bool needs)
  
  Sets whether this widget needs a keyboard.

• unsigned int output () const
  
  Returns if a widget is used for output only.

• Fl_Group *parent () const
  
  Returns a pointer to the parent widget.

• void parent (Fl_Group *p)
  
  Internal use only - “for hacks only”.

• void position (int X, int Y)
  
  Repositions the window or widget.

• void redraw ()
  
  Schedules the drawing of the widget.

• void redraw_label ()
Schedules the drawing of the label.

- `virtual void resize (int x, int y, int w, int h)`
  Changes the size or position of the widget.

- `Fl_Color selection_color () const`
  Gets the selection color.

- `void selection_color (Fl_Color a)`
  Sets the selection color.

- `void set_active ()`
  Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  Marks the value of the widget as changed.

- `void set_output ()`
  Sets a widget to output only.

- `void set_visible ()`
  Makes the widget visible.

- `void set_visible_focus ()`
  Enables keyboard focus navigation with this widget.

- `int shortcut_label () const`
  Returns whether the widget's label uses '&' to indicate shortcuts.

- `void shortcut_label (int value)`
  Sets whether the widget's label uses '&' to indicate shortcuts.

- `virtual void show ()`
  Makes a widget visible.

- `void size (int W, int H)`
  Changes the size of the widget.

- `int take_focus ()`
  Gives the widget the keyboard focus.

- `unsigned int takesevents () const`
  Returns if the widget is able to take events.

- `int test_shortcut ()`
  Returns true if the widget's label contains the entered '&x' shortcut.

- `const char * tooltip () const`
  Gets the current tooltip text.

- `void tooltip (const char *text)`
  Sets the current tooltip text.

- `Fl_Window * top_window () const`
  Returns a pointer to the top-level window for the widget.

- `Fl_Window * top_window_offset (int &xoff, int &yoff) const`
  Finds the x/y offset of the current widget relative to the top-level window.

- `uchar type () const`
  Gets the widget type.

- `void type (uchar t)`
  Sets the widget type.

- `int use_accents_menu ()`
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- `void * user_data () const`
  Gets the user data for this widget.

- `void user_data (Fl_Callback_User_Data *v, bool auto_free)`
  Sets the user data for this widget.

- `void user_data (void *v)`
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window ∗ window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ∼Fl_Widget ()
  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget ∗widget, void ∗data)
  The default callback for all widgets that don't set a callback.
• static unsigned int label_shortcut (const char ∗t)
  Returns the Unicode value of the '&x' shortcut in a given text.
• static int test_shortcut (const char ∗t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

• enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Value_Slider

• void draw () FL_OVERRIDE
  Draws the widget.
Protected Member Functions inherited from Fl_Slider

- void draw (int, int, int, int)
- int handle (int, int, int, int)

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char ∗L)
  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void handle_drag (double newvalue)
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void handle_push ()
  Stores the current value in the previous value.
- void handle_release ()
  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int horizontal () const
  Tells if the valuator is an FL_HORIZONTAL one.
- double previous_value () const
  Gets the previous floating point value before an event changed it.
- void set_value (double v)
  Sets the current floating point value.
- double softclamp (double)
  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()
  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void draw_label () const
  Draws the widget's label at the defined label position.
- void draw_label (int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.
- unsigned int flags () const
  Gets the widget flags mask.
• void h (int v)
  *Internal use only.*
• void set_flag (unsigned int c)
  *Sets a flag in the flags mask.*
• void w (int v)
  *Internal use only.*
• void x (int v)
  *Internal use only.*
• void y (int v)
  *Internal use only.*

The documentation for this class was generated from the following files:
• Fl_Hor_Value_Slider.H
• Fl_Value_Slider.cxx

12.70  **Fl_ICO_Image Class Reference**

The **Fl_ICO_Image** class supports loading, caching, and drawing of Windows icon (.ico) files.

```c
#include <Fl_ICO_Image.H>
```

Inheritance diagram for **Fl_ICO_Image**:

![Inheritance Diagram](inclusion_diagram.png)

**Classes**

• struct IconDirEntry
  *Windows ICONDIRENTRY structure*

**Public Member Functions**

• **Fl_ICO_Image** (const char *filename, int id=-1, const unsigned char *data=NULL, const size_t datasize=0)
  *Loads the named icon image from the given .ico filename or from memory.*
• const IconDirEntry * icondirentry () const
  *Returns the array of idcount() loaded IconDirEntry structures.*
• int idcount () const
  *Returns the number of icons of various resolutions present in the ICO object.*
• ~**Fl_ICO_Image** ()
  *Destructor.*

**Public Member Functions inherited from Fl_BMP_Image**

• **Fl_BMP_Image** (const char *filename)
  *This constructor loads the named BMP image from the given BMP filename.*
• **Fl_BMP_Image** (const char *imagename, const unsigned char *data, const long length=-1)
  *This constructor loads a BMP image from memory.*
Public Member Functions inherited from Fl_RGB_Image

- virtual Fl_SVG_Image * as_svg_image ()
  Returns whether an image is an Fl_SVG_Image or not.
- int cache_h ()
- int cache_w ()
- void color_average (Fl_Color c, float i) FL_OVERRIDE
  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * copy () const
- Fl_Image * copy (int W, int H) const FL_OVERRIDE
  Creates a resized copy of the image.
- void desaturate () FL_OVERRIDE
  The desaturate() method converts an image to grayscale.
- void draw (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE
  Draws the image to the current drawing surface with a bounding box.
- Fl_RGB_Image (const Fl_Pixmap * pxm, Fl_Color bg=FL_GRAY)
  The constructor creates a new RGBA image from the specified Fl_Pixmap.
- Fl_RGB_Image (const uchar * bits, int bits_length, int W, int H, int D, int LD)
  The constructor creates a new image from the specified data.
- Fl_RGB_Image (const uchar * bits, int W, int H, int D=3, int LD=0)
  The constructor creates a new image from the specified data.
- void label (Fl_Menu_Item * m) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget * w) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a widget or menu item.
- virtual void normalize ()
  Makes sure the object is fully initialized.
- void uncache () FL_OVERRIDE
  If the image has been cached for display, delete the cache data.
- ~Fl_RGB_Image () FL_OVERRIDE
  The destructor frees all memory and server resources that are used by the image.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image * as_shared_image ()
  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image * copy () const
  Creates a copy of the image in the same size.
- int count () const
  Returns the number of data values associated with the image.
- int d () const
  Returns the image depth.
- const char * const * data () const
  Returns a pointer to the current image data array.
- int data_h () const
  Returns the height of the image data.
- int data_w () const
  Returns the width of the image data.
- void draw (int X, int Y)
  Draws the image to the current drawing surface.
- int fail () const
Returns a value that is not 0 if there is currently no image available.

- **FL_Image** (int W, int H, int D)
  The constructor creates an empty image with the specified width, height, and depth.

- int h () const
  Returns the current image drawing height in FLTK units.

- void inactive()
  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

- int ld () const
  Returns the current line data size in bytes.

- virtual void release()
  Releases an Fl_Image - the same as 'delete this'.

- virtual void scale (int width, int height, int proportional=1, int can_expand=0)
  Sets the drawing size of the image.

- int w () const
  Returns the current image drawing width in FLTK units.

- virtual ~Fl_Image ()
  The destructor is a virtual method that frees all memory used by the image.

Additional Inherited Members

### Static Public Member Functions inherited from **FL_RGB_Image**

- static size_t max_size()
  Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

- static void max_size (size_t size)
  Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

### Static Public Member Functions inherited from **Fl_Image**

- static Fl_Labeltype define_FL_IMAGE_LABEL()
  Returns the currently used RGB image scaling method.

- static Fl_RGB_Scaling RGB_scaling()
  Sets the RGB image scaling method used for copy(int, int).

- static Fl_RGB_Scaling scaling_algorithm()
  Gets what algorithm is used when resizing a source image to draw it.

- static void scaling_algorithm (Fl_RGB_Scaling algorithm)
  Sets what algorithm is used when resizing a source image to draw it.

### Public Attributes inherited from **FL_RGB_Image**

- int alloc_array
  If non-zero, the object's data array is delete[]'d when deleting the object.

- const uchar * array
  Points to the start of the object's data array.

### Static Public Attributes inherited from **Fl_Image**

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_MEMORY_ACCESS = -4
- static const int ERR_NO_IMAGE = -1
- static bool register_images_done = false
  True after fl_register_images() was called, false before.
Protected Member Functions inherited from Fl_BMP_Image

- void load_bmp_(class Fl_Image_Reader &rdr, int ico_height=0, int ico_width=0)

Protected Member Functions inherited from Fl_Image

- void d (int D)
  *Sets the current image depth.*
- void data (const char ∗const ∗p, int c)
  *Sets the current data pointer and count of pointers in the array.*
- void draw_empty (int X, int Y)
  *The protected method draw_empty() draws a box with an X in it.*
- int draw_scaled (int X, int Y, int W, int H)
  *Draw the image to the current drawing surface rescaled to a given width and height.*
- void h (int H)
  *Sets the height of the image data.*
- void ld (int LD)
  *Sets the current line data size in bytes.*
- void w (int W)
  *Sets the width of the image data.*

Static Protected Member Functions inherited from Fl_Image

- static void labeltype (const Fl_Label ∗lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void measure (const Fl_Label ∗lo, int &lw, int &lh)

12.70.1 Detailed Description

The Fl_ICO_Image class supports loading, caching, and drawing of Windows icon (.ico) files.

12.70.2 Constructor & Destructor Documentation

12.70.2.1 Fl_ICO_Image()

Fl_ICO_Image::Fl_ICO_Image (  
  const char ∗ filename,
  int id = -1,
  const unsigned char ∗ data = NULL,
  const size_t datasize = 0 )

Loads the named icon image from the given .ico filename or from memory.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Name of a .ico file, or of the in-memory image</td>
</tr>
<tr>
<td>id</td>
<td>When id is -1 (default), the highest-resolution icon is loaded; when id 0, load the icon with this ID; when id = -2, load all IconDirEntry structures but no image.</td>
</tr>
<tr>
<td>data</td>
<td>NULL, or in-memory icon data</td>
</tr>
<tr>
<td>datasize</td>
<td>Size in bytes of the data byte array (used when data is not NULL)</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_ICO_Image.H
- Fl_ICO_Image.cxx

Generated by Doxygen
12.71 Fl_Image Class Reference

Base class for image caching, scaling and drawing.
#include <Fl_Image.H>

Inheritance diagram for Fl_Image:

```
Fl_Image
|     |
|     | Fl_Bitmap
|     | Fl_Pixmap
|     | Fl_RGB_Image
|     | Fl_Shared_Image
|     | Fl_Tiled_Image
|     | Fl_XBM_Image
|     | Fl_GIF_Image
|     | Fl_XPM_Image
|     | Fl_BMP_Image
|     | Fl_JPEG_Image
|     | Fl_PNG_Image
|     | Fl_PNM_Image
|     | Fl_SVG_Image
```

Public Member Functions

- virtual class Fl_Shared_Image ∗ as_shared_image ()
  Returns whether an image is an Fl_Shared_Image or not.
- virtual void color_average (Fl_Color c, float i)
  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image ∗ copy () const
  Creates a copy of the image in the same size.
- virtual Fl_Image ∗ copy (int W, int H) const
  Creates a resized copy of the image.
- int count () const
  Returns the number of data values associated with the image.
- int d () const
  Returns the image depth.
- const char ∗ const ∗ data () const
  Returns a pointer to the current image data array.
- int data_h () const
  Returns the height of the image data.
- int data_w () const
  Returns the width of the image data.
- virtual void desaturate ()
  The desaturate() method converts an image to grayscale.
- void draw (int X, int Y)
  Draws the image to the current drawing surface.
- virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
  Draws the image to the current drawing surface with a bounding box.
- int fail () const
  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)
  The constructor creates an empty image with the specified width, height, and depth.
- int h () const
  Returns the current image drawing height in FLTK units.
- void inactive ()
The `inactive()` method calls `color_average(FL_BACKGROUND_COLOR, 0.33f)` to produce an image that appears grayed out.

- **virtual void label (Fl_Menu_Item ∗m)**
  This method is an obsolete way to set the image attribute of a menu item.

- **virtual void label (Fl_Widget ∗w)**
  This method is an obsolete way to set the image attribute of a widget or menu item.

- **int ld () const**
  Returns the current line data size in bytes.

- **virtual void release ()**
  Releases an Fl_Image - the same as 'delete this'.

- **virtual void scale (int width, int height, int proportional=1, int can_expand=0)**
  Sets the drawing size of the image.

- **virtual void uncache ()**
  If the image has been cached for display, delete the cache data.

- **int w () const**
  Returns the current image drawing width in FLTK units.

- **virtual ~Fl_Image ()**
  The destructor is a virtual method that frees all memory used by the image.

**Static Public Member Functions**

- **static Fl_Labeltype define_FL_IMAGE_LABEL ()**
- **static Fl_RGB_Scaling RGB_scaling ()**
  Returns the currently used RGB image scaling method.

- **static void RGB_scaling (Fl_RGB_Scaling)**
  Sets the RGB image scaling method used for copy(int, int).

- **static Fl_RGB_Scaling scaling_algorithm ()**
  Gets what algorithm is used when resizing a source image to draw it.

- **static void scaling_algorithm (Fl_RGB_Scaling algorithm)**
  Sets what algorithm is used when resizing a source image to draw it.

**Static Public Attributes**

- **static const int ERR_FILE_ACCESS = -2**
- **static const int ERR_FORMAT = -3**
- **static const int ERR_MEMORY_ACCESS = -4**
- **static const int ERR_NO_IMAGE = -1**
- **static bool register_images_done = false**
  True after `fl_register_images()` was called, false before.

**Protected Member Functions**

- **void d (int D)**
  Sets the current image depth.

- **void data (const char ∗const ∗p, int c)**
  Sets the current data pointer and count of pointers in the array.

- **void draw_empty (int X, int Y)**
  The protected method `draw_empty()` draws a box with an X in it.

- **int draw_scaled (int X, int Y, int W, int H)**
  Draw the image to the current drawing surface rescaled to a given width and height.

- **void h (int H)**
  Sets the height of the image data.

- **void ld (int LD)**
Sets the current line data size in bytes.

- void w (int W)
  Sets the width of the image data.

Static Protected Member Functions

- static void labeltype (const Fl_Label ∗lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void measure (const Fl_Label ∗lo, int &lw, int &lh)

Friends

- class Fl_Graphics_Driver

### 12.71.1 Detailed Description

Base class for image caching, scaling and drawing.

*Fl_Image* is the base class used for caching, scaling and drawing all kinds of images in FLTK. This class keeps track of common image data such as the pixels, colormap, width, height, and depth. Virtual methods are used to provide type-specific image handling.

Each image possesses two (width, height) pairs:

1. The width and height of the raw image data are returned by *data_w()* and *data_h()*. These values are set when the image is created and remain unchanged.

2. The width and height of the area filled by the image when it gets drawn are returned by *w()* and *h()*.
   These values are equal to *data_w()* and *data_h()* when the image is created and can be changed by the *scale()* member function.

Since the *Fl_Image* class does not support image drawing by itself, calling the *Fl_Image::draw()* method results in a box with an X in it being drawn instead.

### 12.71.2 Constructor & Destructor Documentation

#### 12.71.2.1 Fl_Image()

```cpp
Fl_Image::Fl_Image ( int W, int H, int D )
```

The constructor creates an empty image with the specified width, height, and depth.

The width and height are in pixels. The depth is 0 for bitmaps, 1 for pixmap (colormap) images, and 1 to 4 for color images.

### 12.71.3 Member Function Documentation

#### 12.71.3.1 as_shared_image()

```cpp
virtual class Fl_Shared_Image ∗ Fl_Image::as_shared_image ( ) [inline], [virtual]
```

Returns whether an image is an *Fl_Shared_Image* or not.

This virtual method returns a pointer to an *Fl_Shared_Image* if this object is an instance of *Fl_Shared_Image* or NULL if not. This can be used to detect if a given *Fl_Image* object is a shared image, i.e., derived from *Fl_Shared_Image*.

Since

1.4.0

Reimplemented in *Fl_Shared_Image*. 

---

*Generated by Doxygen*
12.71.3.2  color_average()

void Fl_Image::color_average (  
    Fl_Color c,  
    float i ) [virtual]

The color_average() method averages the colors in the image with the provided FLTK color value.  
The first argument specifies the FLTK color to be used.  
The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color.  
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.  
Reimplemented in Fl_Anim_GIF_Image, Fl_RGB_Image, Fl_Pixmap, Fl_Shared_Image, Fl_SVG_Image, and Fl_Tiled_Image.

12.71.3.3  copy() [1/2]

Fl_Image * Fl_Image::copy ( ) const [inline]

Creates a copy of the image in the same size.  
The new image should be released when you are done with it.  
This does exactly the same as 'Fl_Image::copy(int W, int H) const' where W and H are the width and height of the source image, respectively. This applies also to all subclasses of Fl_Image in the FLTK library.  
The following two copy() calls are equivalent:  
Fl_image *img1 = new Fl_image(...);  
// ...  
Fl_image *img2 = img1->copy();  
Fl_image *img3 = img1->copy(img1->w(), img1->h());  
For details see 'Fl_Image::copy(int w, int h) const'.  
See also  
Fl_Image::release()

Note  
Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your overridden methods of 'Fl_Image::copy() const' and 'Fl_Image::copy(int, int) const' must also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

12.71.3.4  copy() [2/2]

Fl_Image * Fl_Image::copy (  
    int W,  
    int H ) const [virtual]

Creates a resized copy of the image.  
The new image should be released when you are done with it.  
Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator delete for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's.  
The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().  
For the new image the following equations are true:  
\[ w() == \text{data}_w() == W \]  
\[ h() == \text{data}_h() == H \]

Parameters

| in | W/H | Requested width and height of the new image |
Note

The returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your overridden methods of 'Fl_Image::copy() const' and 'Fl_Image::copy(int, int) const' must also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented in Fl_Anim_GIF_Image, Fl_Bitmap, Fl_RGB_Image, Fl_Pixmap, Fl_Shared_Image, Fl_SVG_Image, and Fl_Tiled_Image.

12.71.3.5 count()

int Fl_Image::count ( ) const [inline]
Returns the number of data values associated with the image.
The value will be 0 for images with no associated data, 1 for bitmap and color images, and greater than 2 for pixmap images.
See also
data()

12.71.3.6 d()

int Fl_Image::d ( ) const [inline]
Returns the image depth.
The return value will be 0 for bitmaps, 1 for pixmaps, and 1 to 4 for color images.

12.71.3.7 data() [1/2]

const char *const * Fl_Image::data ( ) const [inline]
Returns a pointer to the current image data array.
There can be 0, 1, or more pointers to actual image data in an image.
Use the count() method to find the size of the data array. You must not dereference the data() pointer if count() equals zero.

Note
data() may return NULL.

Example:
Fl_RGB_Image has exactly one pointer which points at the R, G, B [, A] data array of the image. The total size of this array depends on several attributes like data_w(), data_h(), d() and ld() and is basically data_w() * data_h() * d() but there are exceptions if ld() is non-zero: see description of ld(). Since FLTK 1.4.0 w() and h() are no longer significant for the image data size if scale() has been called on the image to set a different display size.
Other image types have different numbers and types of data pointers which are implementation details and not documented here.
See also

count(), w(), h(), data_w(), data_h(), d(), ld()

12.71.3.8 data() [2/2]

void Fl_Image::data ( 
    const char *const * p,
    int c ) [inline], [protected]
Sets the current data pointer and count of pointers in the array.
There can be 0, 1, or more pointers to actual image data in an image.
12.71 Fl_Image Class Reference

See also

const char* const* data(), count(), w(), h(), data_w(), data_h(), d(), ld()

12.71.3.9 desaturate()

void Fl_Image::desaturate () [virtual]

The desaturate() method converts an image to grayscale.
If the image contains an alpha channel (depth = 4), the alpha channel is preserved.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original
image data in memory.
Reimplemented in Fl_Anim_GIF_Image, Fl_RGB_Image, Fl_Pixmap, Fl_Shared_Image, Fl_SVG_Image, and
Fl_Tiled_Image.

12.71.3.10 draw() [1/2]

void Fl_Image::draw (  
  int X,  
  int Y ) [inline]

Draws the image to the current drawing surface.

Parameters

X, Y  specify the upper-lefthand corner of the image.

12.71.3.11 draw() [2/2]

void Fl_Image::draw (  
  int X,  
  int Y,  
  int W,  
  int H,  
  int cx = 0,  
  int cy = 0 ) [virtual]

Draws the image to the current drawing surface with a bounding box.
Arguments X, Y, W, H specify a bounding box for the image, with the origin (upper-left corner) of the image offset
by the cx and cy arguments.
In other words: fl_push_clip(X,Y,W,H) is applied, the image is drawn with its upper-left corner at
X-cx,Y-cy and its own width and height, fl_pop_clip() is applied.
Reimplemented in Fl_Anim_GIF_Image, Fl_Bitmap, Fl_RGB_Image, Fl_Pixmap, Fl_Shared_Image, Fl_SVG_Image,
and Fl_Tiled_Image.

12.71.3.12 draw_empty()

void Fl_Image::draw_empty (  
  int X,  
  int Y ) [protected]

The protected method draw_empty() draws a box with an X in it.
It can be used to draw any image that lacks image data.

12.71.3.13 draw_scaled()

int Fl_Image::draw_scaled (  
  int X,  
  int Y,  
  int W,  
  int H ) [protected]

Draw the image to the current drawing surface rescaled to a given width and height.
Intended for internal use by the FLTK library.

Parameters

<table>
<thead>
<tr>
<th>X,Y</th>
<th>position of the image's top-left</th>
</tr>
</thead>
<tbody>
<tr>
<td>W,H</td>
<td>width and height for the drawn image</td>
</tr>
</tbody>
</table>

Returns

1

Deprecated Only for API compatibility with FLTK 1.3.4.

12.71.3.14 fail()

int Fl_Image::fail () const

Returns a value that is not 0 if there is currently no image available.

Example use:

```c
// [...]  
Fl_Box box(X, Y, W, H);  
Fl_JPEG_Image jpg("/tmp/foo.jpg");  
switch (jpg.fail()) {  
  case Fl_Image::ERR_NO_IMAGE:  
    case Fl_Image::ERR_FILE_ACCESS:  
      fl_alert("/tmp/foo.jpg: %s", strerror(errno)); // shows actual os error to user  
      exit(1);  
    case Fl_Image::ERR_FORMAT:  
      fl_alert("/tmp/foo.jpg: couldn't decode image");  
      exit(1);  
  }  
box.image(jpg);  
```

Returns

Image load failure if non-zero

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>the image was loaded successfully</td>
</tr>
<tr>
<td>ERR_NO_IMAGE</td>
<td>no image was found</td>
</tr>
<tr>
<td>ERR_FILE_ACCESS</td>
<td>there was a file access related error (errno should be set)</td>
</tr>
<tr>
<td>ERR_FORMAT</td>
<td>image decoding failed</td>
</tr>
<tr>
<td>ERR_MEMORY_ACCESS</td>
<td>image decoder tried to access memory outside of given memory block</td>
</tr>
</tbody>
</table>

12.71.3.15 h() [1/2]

int Fl_Image::h () const [inline]

Returns the current image drawing height in FLTK units.

The values of h() and data_h() are identical unless scale() has been called after which they may differ.

12.71.3.16 h() [2/2]

void Fl_Image::h (  
  int H ) [inline], [protected]

Sets the height of the image data.

This protected function sets both image heights: the height of the image data returned by data_h() and the image drawing height in FLTK units returned by h().

---

Generated by Doxygen
12.71.3.17 \textbf{inactive()}

```cpp
void Fl_Image::inactive ( ) [inline]
```

The \texttt{inactive()} method calls \texttt{color\_average(FL\_BACKGROUND\_COLOR, 0.33f)} to produce an image that appears grayed out.

An internal copy is made of the original image before changes are applied, to avoid modifying the original image.

\textbf{Note}

The RGB color of \texttt{FL\_BACKGROUND\_COLOR} may change when the connection to the display is made. See \texttt{fl\_open\_display()}.

12.71.3.18 \textbf{label() [1/2]}

```cpp
void Fl_Image::label ( Fl_Menu_Item ∗ m ) [virtual]
```

This method is an obsolete way to set the image attribute of a menu item.

\textbf{Deprecated} Please use \texttt{Fl\_Menu\_Item::image()} instead.

Reimplemented in \texttt{Fl\_Bitmap}, \texttt{Fl\_RGB\_Image}, and \texttt{Fl\_Pixmap}.

12.71.3.19 \textbf{label() [2/2]}

```cpp
void Fl_Image::label ( Fl_Widget ∗ widget ) [virtual]
```

This method is an obsolete way to set the image attribute of a widget or menu item.

\textbf{Deprecated} Please use \texttt{Fl\_Widget::image()} or \texttt{Fl\_Widget::deimage()} instead.

Reimplemented in \texttt{Fl\_Bitmap}, \texttt{Fl\_RGB\_Image}, and \texttt{Fl\_Pixmap}.

12.71.3.20 \textbf{ld() [1/2]}

```cpp
int Fl_Image::ld ( ) const [inline]
```

Returns the current line data size in bytes.

See also

\texttt{ld(int)}

12.71.3.21 \textbf{ld() [2/2]}

```cpp
void Fl_Image::ld ( int LD ) [inline], [protected]
```

Sets the current line data size in bytes.

Color images may contain extra data (padding) that is included after every line of color image data and is normally not present.

If \texttt{LD} is zero, then line data size is assumed to be \texttt{data\_w()} \texttt{* d()} bytes.

If \texttt{LD} is non-zero, then it must be positive and larger than \texttt{data\_w()} \texttt{* d()} to account for the extra data per line.

12.71.3.22 \textbf{release()}

```cpp
virtual void Fl_Image::release ( ) [inline], [virtual]
```

Releases an \texttt{Fl\_Image} - the same as 'delete this'.

This virtual method is for almost all image classes the same as calling \texttt{delete image;}

where \texttt{image} is an \texttt{Fl\_Image ∗ pointer}.

However, for subclass \texttt{Fl\_Shared\_Image} and its subclasses this virtual method is reimplemented and maintains shared images.

This virtual method makes it possible to \texttt{destroy} all image types in the same way by calling...
Reasoning: If you have an `Fl_Image` base class pointer and don’t know if the object is one of the class `Fl_Shared_Image` or any other subclass of `Fl_Image` (for instance `Fl_RGB_Image`) then you can't just use operator delete since this is not appropriate for `Fl_Shared_Image` objects. The virtual method `release()` handles this properly.

Since 1.4.0 in the base class `Fl_Image` and virtual in `Fl_Shared_Image`.

Reimplemented in `Fl_Shared_Image`.

### 12.71.3.23 RGB_scaling()

```cpp
void Fl_Image::RGB_scaling (Fl_RGB_Scaling method) [static]
```

Sets the RGB image scaling method used for `copy(int, int)`. Applies to all RGB images, defaults to `FL_RGB_SCALING_NEAREST`.

### 12.71.3.24 scale()

```cpp
void Fl_Image::scale (int width, int height, int proportional = 1, int can_expand = 0) [virtual]
```

Sets the drawing size of the image. This function controls the values returned by member functions `w()` and `h()` which in turn control how the image is drawn: the full image data (whose size is given by `data_w()` and `data_h()` are drawn scaled to an area of the drawing surface sized at `w()` x `h()` FLTK units. This can make a difference if the drawing surface has more than 1 pixel per FLTK unit because the image can be drawn at the full resolution of the drawing surface. Examples of such drawing surfaces: HiDPI displays, laser printers, PostScript files, PDF printers.

**Parameters**

<table>
<thead>
<tr>
<th>width, height</th>
<th>maximum values, in FLTK units, that <code>w()</code> and <code>h()</code> should return</th>
</tr>
</thead>
<tbody>
<tr>
<td>proportional</td>
<td>if not null, keep the values returned by <code>w()</code> and <code>h()</code> proportional to <code>data_w()</code> and <code>data_h()</code></td>
</tr>
<tr>
<td>can_expand</td>
<td>if null, the values returned by <code>w()</code> and <code>h()</code> will not be larger than <code>data_w()</code> and <code>data_h()</code>, respectively</td>
</tr>
</tbody>
</table>

**Note**

This function generally changes the values returned by the `w()` and `h()` member functions. In contrast, the values returned by `data_w()` and `data_h()` remain unchanged.

**Version**

1.4 (1.3.4 and FL_ABI_VERSION for `Fl_Shared_Image` only)

**Example code: scale an image to fit in a box**

```cpp
Fl_Box *b = ... // a box
Fl_Image *img = new Fl_PNG_Image("/path/to/picture.png"); // read a picture file
img->scale(b->w(), b->h()); // set the drawing size of the image to the size of the box keeping its aspect ratio
b->image(img); // use the image as the box image
```

### 12.71.3.25 scaling_algorithm()

```cpp
static void Fl_Image::scaling_algorithm (Fl_RGB_Scaling algorithm) [inline], [static]
```

Sets what algorithm is used when resizing a source image to draw it. The default algorithm is `FL_RGB_SCALING_BILINEAR`. Drawing an `Fl_Image` is sometimes performed by first resizing the source image and then drawing the resized copy. This occurs, e.g., when drawing to screen under X11.
without Xrender support after having called scale(). This function controls what method is used when the image to
be resized is an Fl_RGB_Image.

Version

1.4

### 12.71.3.26 uncache()

```cpp
void Fl_Image::uncache () [virtual]
```

If the image has been cached for display, delete the cache data.
This allows you to change the data used for the image and then redraw it without recreating an image object.
Reimplemented in Fl_Anim_GIF_Image, Fl_Bitmap, Fl_RGB_Image, Fl_Pixmap, and Fl_Shared_Image.

### 12.71.3.27 w() [1/2]

```cpp
int Fl_Image::w () const [inline]
```

Returns the current image drawing width in FLTK units.
The values of w() and data_w() are identical unless scaled() has been called after which they may differ.

### 12.71.3.28 w() [2/2]

```cpp
void Fl_Image::w (int \ W\ ) [inline], [protected]
```

Sets the width of the image data.
This protected function sets both image widths: the width of the image data returned by data_w() and the image
drawing width in FLTK units returned by w().

The documentation for this class was generated from the following files:

- Fl_Image.H
- Fl_Image.cxx

---

### 12.72 Fl_Image_Reader Class Reference

#### Public Member Functions

- int error () const
- const char * name () const
- int open (const char * filename)
- int open (const char * imagename, const unsigned char * data)
- int open (const char * imagename, const unsigned char * data, const size_t datasize)
- unsigned char read_byte ()
- unsigned int read_dword ()
- int read_long ()
- unsigned short read_word ()
- void seek (unsigned int n)
- void skip (unsigned int n)
- long tell () const

The documentation for this class was generated from the following files:

- Fl_Image_Reader.h
- Fl_Image_Reader.cxx
12.73  Fl_Image_Surface Class Reference

Directs all graphics requests to an Fl_Image.
#include <Fl_Image_Surface.H>

Inheritance diagram for Fl_Image_Surface:

```
Fl_Surface_Device
   ↓
Fl_Widget_Surface
   ↓
Fl_Image_Surface
```

Public Member Functions

- **Fl_Image_Surface** (int w, int h, int high_res=0, Fl_Offscreen off=0)
  Constructor with optional high resolution.
- **Fl_Shared_Image** * highres_image ()
  Returns a possibly high resolution image made of all drawings sent to the Fl_Image_Surface object.
- **Fl_RGB_Image** * image ()
  Returns a depth-3 image made of all drawings sent to the Fl_Image_Surface object.
- bool is_current () FL_OVERRIDE
  Is this surface the current drawing surface?
- void mask (const Fl_RGB_Image *)
  Defines a mask applied to drawings made after use of this function.
- Fl_Offscreen offscreen ()
  Returns the Fl_Offscreen object associated to the image surface.
- void origin (int *x, int *y) FL_OVERRIDE
  Computes the coordinates of the current origin of graphics functions.
- void origin (int x, int y) FL_OVERRIDE
  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- int printable_rect (int *w, int *h) FL_OVERRIDE
  Computes the width and height of the drawable area of the drawing surface.
- void rescale ()
  Adapts the Fl_Image_Surface object to the new value of the GUI scale factor.
- void set_current () FL_OVERRIDE
  Make this surface the current drawing surface.
- ~Fl_Image_Surface ()
  The destructor.

Public Member Functions inherited from Fl_Widget_Surface

- void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)
  Draws the widget on the drawing surface.
- void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)
  Draws a window with its title bar and frame if any.
- void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)
  Draws a rectangular part of an on-screen window.
Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver * driver ()
  Returns the graphics driver of this drawing surface.
- virtual ~Fl_Surface_Device ()
  The destructor.

Protected Member Functions

- void translate (int x, int y) FL_OVERRIDE
  Translates the current graphics origin accounting for the current rotation.
- void untranslated () FL_OVERRIDE
  Undoes the effect of a previous translate() call.

Protected Member Functions inherited from Fl_Widget_Surface

- Fl_Widget_Surface (Fl_Graphics_Driver *d)
  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- void driver (Fl_Graphics_Driver *graphics_driver)
  Sets the graphics driver of this drawing surface.
- virtual void end_current ()
  FLTK calls this each time a surface ceases to be the current drawing surface.
- Fl_Surface_Device (Fl_Graphics_Driver *graphics_driver)
  Constructor that sets the graphics driver to use for the created surface.

Friends

- class Fl_Graphics_Driver

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()
  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device *new_current)
  Pushes new_current on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()
  The current drawing surface.

Protected Attributes inherited from Fl_Widget_Surface

- int x_offset
  horizontal offset to the origin of graphics coordinates
- int y_offset
  vertical offset to the origin of graphics coordinates
12.73.1 Detailed Description

Directs all graphics requests to an Fl_Image.

After creation of an Fl_Image_Surface object, make it the current drawing surface calling Fl_Surface_Device::push_current(), and all subsequent graphics requests will be recorded in the image. It's possible to draw widgets (using Fl_Image_Surface::draw()) or to use any of the Drawing functions or the Color & Font functions. Finally, call image() on the object to obtain a newly allocated Fl_RGB_Image object. Fl_Gl_Window objects can be drawn in the image as well.

Usage example:

```c
// this is the widget that you want to draw into an image
Fl_Widget *g = ...;

// create an Fl_Image_Surface object
Fl_Image_Surface *image_surface = new Fl_Image_Surface(g->w(), g->h());

// direct all further graphics requests to the image
Fl_Surface_Device::push_current(image_surface);

// draw a white background
fl_color(FL_WHITE);
fl_rectf(0, 0, g->w(), g->h());

// direct the g widget in the image
image_surface->draw(g);

// get the resulting image
Fl_RGB_Image* image = image_surface->image();

// direct graphics requests back to their previous destination
Fl_Surface_Device::pop_current();

// delete the image_surface object, but not the image itself
delete image_surface;
```

12.73.2 Constructor & Destructor Documentation

12.73.2.1 Fl_Image_Surface()

Fl_Image_Surface::Fl_Image_Surface (int w, int h, int high_res = 0, Fl_Offscreen off = 0)

Constructor with optional high resolution.

Parameters

| w,h | Width and height of the resulting image. The value of the high_res parameter controls whether w and h are interpreted as pixels or FLTK units. |
| high_res | If zero, the created image surface is sized at w * h pixels. If non-zero, the pixel size of the created image surface depends on the value of the display scale factor (see Fl::screen_scale(int)): the resulting image has the same number of pixels as an area of the display of size w * h expressed in FLTK units. |
| off | If not null, the image surface is constructed around a pre-existing Fl_Offscreen. The caller is responsible for both construction and destruction of this Fl_Offscreen object. Is mostly intended for internal use by FLTK. |

Version

1.3.4 (1.3.3 without the highres parameter)

12.73.3 Member Function Documentation

12.73.3.1 highres_image()

Fl_Shared_Image * Fl_Image_Surface::highres_image ()

Returns a possibly high resolution image made of all drawings sent to the Fl_Image_Surface object.
The **Fl_Image_Surface** object should have been constructed with `Fl_Image_Surface(W, H, 1)`. The returned **Fl_Shared_Image** object is scaled to a size of WxH FLTK units and may have a pixel size larger than these values. The returned object should be deallocated with `Fl_Shared_Image::release()` after use.

**Deprecated** Use `image()` instead.

**Version**

1.4 (1.3.4 for MacOS platform only)

### 12.73.3.2 image()

```c
Fl_RGB_Image * Fl_Image_Surface::image ()
```

Returns a depth-3 image made of all drawings sent to the **Fl_Image_Surface** object. The returned object contains its own copy of the RGB data; the caller is responsible for deleting it.

See also

`Fl_Image_Surface::mask(Fl_RGB_Image*)`

### 12.73.3.3 is_current()

```c
bool Fl_Image_Surface::is_current () [virtual]
```

Is this surface the current drawing surface?

Reimplemented from **Fl_Surface_Device**.

### 12.73.3.4 mask()

```c
void Fl_Image_Surface::mask (const Fl_RGB_Image * mask)
```

Defines a mask applied to drawings made after use of this function.

The mask is an **Fl_RGB_Image** made of a white scene drawn on a solid black background; the drawable part of the image surface is reduced to the white areas of the mask after this member function gets called. If necessary, the **mask** image is internally replaced by a copy resized to the surface's pixel size. Overall, the image returned by `Fl_Image_Surface::image()` contains all drawings made until the `mask()` method assigned a mask, at which point subsequent drawing operations to the image surface were passed through the white areas of the mask. On some platforms, shades of gray in the mask image control the blending of foreground and background pixels; mask pixels closer in color to white produce image pixels closer to the image surface pixel, those closer to black produce image pixels closer to what the image surface pixel was before the call to `mask()`.

The mask is easily constructed using an **Fl_Image_Surface** object, drawing white areas on a black background there, and calling `Fl_Image_Surface::image()`.

**Parameters**

| **mask** | A depth-3 image determining the drawable areas of the image surface. The **mask** object is not used after return from this member function. |

**Note**

- The image surface must not be the current drawing surface when this function gets called.
- The mask can have any size but is best when it has the size of the image surface.
- It's possible to use several masks in succession on the same image surface provided member function `Fl_Image_Surface::image()` is called between successive calls to `Fl_Image_Surface::mask(const Fl_RGB_Image*)`.

**Example of procedure to construct a masked image:**

```c
int W = ..., H = ...; // width and height of the image under construction
Fl_Image_Surface *surf = new Fl_Image_Surface(W, H, 1);
// first, construct the mask
Fl_Surface_Device::push_current(surf);
fl_color(FL_BLACK); // draw a black background
```
```
fl_rectf(0, 0, W, H);
fl_color(FL_WHITE); // next, draw in white what the mask should not filter out
fl_pie(0, 0, W, H, 0, 360); // here, an ellipse with axes lengths WxH
Fl_RGB_Image *mask = surf->image(); // get the mask
// second, draw the image background
fl_color(FL_YELLOW); // here, draw a yellow background
fl_rectf(0, 0, W, H);
// third, apply the mask
Fl_Surface_Device::pop_current();
surf->mask(mask);
delete mask; // the mask image can be safely deleted at this point
Fl_Surface_Device::push_current(surf);
// fourth, draw the image foreground, part of which will be filtered out by the mask
surf->draw(widget, 0, 0); // here the foreground is a drawn widget
Fl_RGB_Image *masked_image = surf->image();
// Only the part of the foreground, here a drawn widget, that has not been
// filtered out by the mask, here the white ellipse, is in masked_image;
// the background, here solid yellow, shows up in the remaining areas of masked_image.
Fl_Surface_Device::pop_current();
delete surf;
```

Since

1.4.0

### 12.73.3.5 `offscreen()`

**Fl_Offscreen Fl_Image_Surface::offscreen ()**

Returns the Fl_Offscreen object associated to the image surface.

The returned Fl_Offscreen object is deleted when the Fl_Image_Surface object is deleted, unless the Fl_Image_Surface was constructed with non-null Fl_Offscreen argument.

### 12.73.3.6 `origin()` [1/2]

```c
void Fl_Image_Surface::origin ( 
    int */ x,
    int */ y ) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

**Parameters**

| out x,y | If non-null, +x and +y are set to the horizontal and vertical coordinates of the graphics origin. |

Reimplemented from Fl_Widget_Surface.

### 12.73.3.7 `origin()` [2/2]

```c
void Fl_Image_Surface::origin ( 
    int x,
    int y ) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.

Arguments should be expressed relatively to the result of a previous `printable_rect()` call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive `origin()` calls don't combine their effects. Origin() calls are not affected by `rotate()` calls (for classes derived from Fl_Paged_Device).

**Parameters**

| in x,y | Horizontal and vertical positions in the drawing surface of the desired origin of graphics. |

Reimplemented from Fl_Widget_Surface.
12.73.3.8 printable_rect()

int Fl_Image_Surface::printable_rect (  
    int * w,  
    int * h )  [virtual]  
Computes the width and height of the drawable area of the drawing surface.  
Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.  

Returns  
0 if OK, non-zero if any error  
Reimplemented from Fl_Widget_Surface.

12.73.3.9 rescale()

void Fl_Image_Surface::rescale ( )  
Adapts the Fl_Image_Surface object to the new value of the GUI scale factor.  
The Fl_Image_Surface object must not be the current drawing surface. This function is useful only for an object constructed with non-zero high_res parameter.  

Version  
1.4

12.73.3.10 set_current()

void Fl_Image_Surface::set_current (  
    void )  [virtual]  
Make this surface the current drawing surface.  
This surface will receive all future graphics requests. Starting from FLTK 1.4.0, the preferred API to change the current drawing surface is Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current().  

Note  
It's recommended to use this function only as follows:  
• The current drawing surface is the display;  
• make current another surface, e.g., an Fl_Printer or an Fl_Image_Surface object, calling set_current() on this object;  
• draw to that surface;  
• make the display current again with Fl_Display_Device::display_device() -> set_current();. Don't do any other call to set_current() before this one.  

Other scenarios of drawing surface changes should be performed via Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current().  
Reimplemented from Fl_Surface_Device.

12.73.3.11 translate()

void Fl_Image_Surface::translate (  
    int x,  
    int y )  [protected], [virtual]  
Translates the current graphics origin accounting for the current rotation.  
Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.  
Reimplemented from Fl_Widget_Surface.
12.73.3.12 untranslate()

void Fl_Image_Surface::untranslate ( 
    void ) [protected], [virtual]

Undoes the effect of a previous translate() call.
Reimplemented from Fl_Widget_Surface.
The documentation for this class was generated from the following files:

• Fl_Image_Surface.H
• Fl_Image_Surface.cxx

12.74 Fl_Input Class Reference

This is the FLTK text input widget.
#include <Fl_Input.H>

Inheritance diagram for Fl_Input:

Public Member Functions

• Fl_Input (int, int, int, int, const char ∗=0)
  Creates a new Fl_Input widget using the given position, size, and label string.
• int handle (int) FL_OVERRIDE
  Handles the specified event.

Public Member Functions inherited from Fl_Input_

• int append (const char ∗t, int l=0, char keep_selection=0)
  Append text at the end.
• bool can_redo () const
  Check if there is a redo action available.
• bool can_undo () const
  Check if the last operation can be undone.
• int copy (int clipboard)
  Put the current selection into the clipboard.
• int copy_cuts ()
  Copies the yank buffer to the clipboard.
• Fl_Color cursor_color () const
  Gets the color of the cursor.
• void cursor_color (Fl_Color n)
  Sets the color of the cursor.
• int cut ()
  Deletes the current selection.
• int cut (int a, int b)
  Deletes all characters between index a and b.
• int cut (int n)
  Deletes the next n bytes rounded to characters before or after the cursor.
• double dvalue () const
Returns the widget text interpreted as a floating point number.

- `Fl_Input_ (int, int, int, int, const char * = 0)`
  Creates a new `Fl_Input_` widget.

- `unsigned int index (int i) const`
  Returns the character at index `i`.

- `int input_type () const`
  Gets the input field type.

- `void input_type (int t)`
  Sets the input field type.

- `int insert (const char * t, int l = 0)`
  Inserts text at the cursor position.

- `int insert_position () const`
  Gets the position of the text cursor.

- `int insert_position (int p)`
  Sets the text cursor position and mark.

- `int insert_position (int p, int m)`
  Sets the index for the cursor and mark.

- `int ivalue () const`
  Returns the widget text interpreted as a signed integer.

- `int mark () const`
  Gets the current selection mark.

- `int mark (int m)`
  Sets the current selection mark.

- `int maximum_size () const`
  Gets the maximum length of the input field in characters.

- `void maximum_size (int m)`
  Sets the maximum length of the input field in characters.

- `int position () const`
  Gets the read-only state of the input field.

- `void position (int p)`
  Sets the read-only state of the input field.

- `int redo ()`
  Redo previous undo operation.

- `int replace (int b, int e, const char * text, int ilen = 0)`
  Deletes text from `b` to `e` and inserts the new string `text`.

- `void resize (int, int, int, int) FL_OVERRIDE`
  Changes the size of the widget.

- `int shortcut () const`
  Return the shortcut key associated with this widget.

- `void shortcut (int s)`
  Sets the shortcut key associated with this widget.

- `int size () const`
  Returns the number of bytes in `value()`.

- `void size (int W, int H)`
  Sets the width and height of this widget.

- `int static_value (const char * s)`
  Changes the widget text.

- `int static_value (const char * s, int)`
  Changes the widget text.
Changes the widget text.

- **int tab_nav () const**  
  Gets whether the Tab key causes focus navigation in multiline input fields or not.

- **void tab_nav (int val)**  
  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.

- **Fl_Color textcolor () const**  
  Gets the color of the text in the input field.

- **void textcolor (Fl_Color n)**  
  Sets the color of the text in the input field.

- **Fl_Font textfont () const**  
  Gets the font of the text in the input field.

- **void textfont (Fl_Font s)**  
  Sets the font of the text in the input field.

- **Fl_Fontsize textsize () const**  
  Gets the size of the text in the input field.

- **void textsize (Fl_Fontsize s)**  
  Sets the size of the text in the input field.

- **int undo ()**  
  Undoes previous changes to the text buffer.

- **const char ∗ value () const**  
  Returns the text displayed in the widget.

- **int value (const char ∗)***  
  Changes the widget text.

- **int value (const char ∗, int)**  
  Changes the widget text.

- **int value (double value)***  
  Changes the widget text to a floating point number ("%g").

- **int value (int value)**  
  Changes the widget text to a signed integer number.

- **int wrap () const**  
  Gets the word wrapping state of the input field.

- **void wrap (int b)**  
  Sets the word wrapping state of the input field.

- **~Fl_Input_ ()**  
  Destroys the widget.

**Public Member Functions inherited from Fl_Widget**

- **void _clear_fullscreen ()**
- **void _set_fullscreen ()**
- **void activate ()**  
  Activates the widget.

- **unsigned int active () const**  
  Returns whether the widget is active.

- **int active_r () const**  
  Returns whether the widget and all of its parents are active.

- **Fl_Align align () const**  
  Gets the label alignment.

- **void align (Fl_Align alignment)**  
  Sets the label alignment.

- **long argument () const**
Gets the current user data (long) argument that is passed to the callback function.

- `void argument (long v)`
  Sets the current user data (long) argument that is passed to the callback function.

- `virtual class Fl_Gl_Window ∗ as_gl_window ()`
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- `virtual class Fl_Gl_Window const ∗ as_gl_window () const`
- `virtual Fl_Group ∗ as_group ()`
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- `virtual Fl_Group const ∗ as_group () const`
- `virtual Fl_Window ∗ as_window ()`
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- `virtual Fl_Window const ∗ as_window () const`
- `void bind_deimage (Fl_Image ∗ img)`
  Sets the image to use as part of the widget label when in the inactive state.

- `void bind_deimage (int f)`
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- `void bind_image (Fl_Image ∗ img)`
  Sets the image to use as part of the widget label when in the active state.

- `void bind_image (int f)`
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- `Fl_Boxtype box () const`
  Gets the box type of the widget.

- `void box (Fl_Boxtype new_box)`
  Sets the box type for the widget.

- `Fl_Callback_p callback () const`
  Gets the current callback function for the widget.

- `void callback (Fl_Callback ∗ cb)`
  Sets the current callback function for the widget.

- `void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)`
  Sets the current callback function and managed user data for the widget.

- `void callback (Fl_Callback ∗ cb, void ∗ p)`
  Sets the current callback function and data for the widget.

- `void callback (Fl_Callback0 ∗ cb)`
  Sets the current callback function for the widget.

- `void callback (Fl_Callback1 ∗ cb, long p=0)`
  Sets the current callback function for the widget.

- `unsigned int changed () const`
  Checks if the widget value changed since the last callback.

- `void clear_active ()`
  Marks the widget as inactive without sending events or changing focus.

- `void clear_changed ()`
  Marks the value of the widget as unchanged.

- `void clear_damage (uchar c=0)`
  Clears or sets the damage flags.

- `void clear_output ()`
  Sets a widget to accept input.

- `void clear_visible ()`
  Hides the widget.

- `void clear_visible_focus ()`
  Disables keyboard focus navigation with this widget.

- `Fl_Color color () const`
  Gets the current color of the widget.
Gets the background color of the widget.

- **void color (Fl_Color bg)**
  - Sets the background color of the widget.
- **void color (Fl_Color bg, Fl_Color sel)**
  - Sets the background and selection color of the widget.
- **Fl_Color color2 () const**
  - For back compatibility only.
- **void color2 (unsigned a)**
  - For back compatibility only.
- **int contains (const Fl_Widget *w) const**
  - Checks if w is a child of this widget.
- **void copy_label (const char *new_label)**
  - Sets the current label.
- **void copy_tooltip (const char *text)**
  - Sets the current tooltip text.
- **uchar damage () const**
  - Returns non-zero if draw() needs to be called.
- **void damage (uchar c)**
  - Sets the damage bits for the widget.
- **void damage (uchar c, int x, int y, int w, int h)**
  - Sets the damage bits for an area inside the widget.
- **int damage_resize (int, int, int, int)**
  - Internal use only.
- **void deactivate ()**
  - Deactivates the widget.
- **Fl_Image * deimage ()**
  - Gets the image that is used as part of the widget label when in the inactive state.
- **const Fl_Image * deimage () const**
  - Gets the image that is used as part of the widget label when in the inactive state.
- **void deimage (Fl_Image &img)**
  - Sets the image to use as part of the widget label when in the inactive state.
- **void deimage (Fl_Image *img)**
  - Sets the image to use as part of the widget label when in the inactive state.
- **int deimage_bound () const**
  - Returns whether the inactive image is managed by the widget.
- **void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  - Calls the widget callback function with default arguments.
- **void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  - Calls the widget callback function with arbitrary arguments.
- **void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  - Calls the widget callback function with arbitrary arguments.
- **void draw_label (int, int, int, int, Fl_Align) const**
  - Draws the label in an arbitrary bounding box with an arbitrary alignment.
- **int h () const**
  - Gets the widget height.
- **virtual void hide ()**
  - Makes a widget invisible.
- **Fl_Image * image ()**
  - Gets the image that is used as part of the widget label when in the active state.
- **const Fl_Image * image () const**
  - Gets the image that is used as part of the widget label when in the active state.
- **void image (Fl_Image &img)**
  Sets the image to use as part of the widget label when in the active state.
- **void image (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the active state.
- **int image_bound () const**
  Returns whether the image is managed by the widget.
- **int inside (const Fl_Widget *wgt) const**
  Checks if this widget is a child of wgt.
- **int is_label_copied () const**
  Returns whether the current label was assigned with copy_label().
- **const char * label () const**
  Gets the current label text.
- **void label (const char *text)**
  Sets the current label pointer.
- **void label (Fl_Labeltype a, const char *b)**
  Shortcut to set the label text and type in one call.
- **Fl_Color labelcolor () const**
  Gets the label color.
- **void labelcolor (Fl_Color c)**
  Sets the label color.
- **Fl_Font labelfont () const**
  Gets the font to use.
- **void labelfont (Fl_Font f)**
  Sets the font to use.
- **Fl_Fontsize labelsize () const**
  Gets the font size in pixels.
- **void labelsize (Fl_Fontsize pix)**
  Sets the font size in pixels.
- **Fl_Labeltype labeltype () const**
  Gets the label type.
- **void labeltype (Fl_Labeltype a)**
  Sets the label type.
- **void measure_label (int &ww, int &hh) const**
  Sets width ww and height hh accordingly with the label size.
- **bool needs_keyboard () const**
  Returns whether this widget needs a keyboard.
- **void needs_keyboard (bool needs)**
  Sets whether this widget needs a keyboard.
- **unsigned int output () const**
  Returns if a widget is used for output only.
- **Fl_Group * parent () const**
  Returns a pointer to the parent widget.
- **void parent (Fl_Group *p)**
  Internal use only - “for hacks only”.
- **void position (int X, int Y)**
  Repositions the window or widget.
- **void redraw ()**
  Schedules the drawing of the widget.
- **void redraw_label ()**
  Schedules the drawing of the label.
- **Fl_Color selection_color () const**
• **void** `selection_color (Fl_Color a)`
  
  Gets the selection color.

• **void** `set_active ()`
  
  Marks the widget as active without sending events or changing focus.

• **void** `set_changed ()`
  
  Marks the value of the widget as changed.

• **void** `set_output ()`
  
  Sets a widget to output only.

• **void** `set_visible ()`
  
  Makes the widget visible.

• **void** `set_visible_focus ()`
  
  Enables keyboard focus navigation with this widget.

• **int** `shortcut_label () const`
  
  Returns whether the widget's label uses '&' to indicate shortcuts.

• **void** `shortcut_label (int value)`
  
  Sets whether the widget's label uses '&' to indicate shortcuts.

• **virtual** **void** `show ()`
  
  Makes a widget visible.

• **void** `size (int W, int H)`
  
  Changes the size of the widget.

• **int** `take_focus ()`
  
  Gives the widget the keyboard focus.

• **unsigned int** `takesevents () const`
  
  Returns if the widget is able to take events.

• **int** `testShortcut ()`
  
  Returns true if the widget's label contains the entered 'x' shortcut.

• **const char** ∗`tooltip () const`
  
  Gets the current tooltip text.

• **void** `tooltip (const char ∗text)`
  
  Sets the current tooltip text.

• **Fl_Window** ∗`top_window () const`
  
  Returns a pointer to the top-level window for the widget.

• **Fl_Window** ∗`top_window_offset (int &xoff, int &yoff) const`
  
  Finds the x/y offset of the current widget relative to the top-level window.

• **uchar** `type () const`
  
  Gets the widget type.

• **void** `type (uchar t)`
  
  Sets the widget type.

• **int** `use_accents_menu ()`
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• **void** `*user_data () const`
  
  Gets the user data for this widget.

• **void** `user_data (Fl_Callback_User_Data ∗v, bool auto_free)`
  
  Sets the user data for this widget.

• **void** `user_data (void ∗v)`
  
  Sets the user data for this widget.

• **unsigned int** `visible () const`
  
  Returns whether a widget is visible.

• **unsigned int** `visible_focus () const`
  
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.

Static Public Attributes

• static const char * copy_menu_text = "Copy"
  [this text may be customized at run-time]
• static const char * cut_menu_text = "Cut"
  [this text may be customized at run-time]
• static const char * paste_menu_text = "Paste"
  [this text may be customized at run-time]

Protected Member Functions

• void draw () FL_OVERRIDE
  Draws the widget.
• int handle_key ()
  Handles a keystroke.
• int handle_rmb ()
  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input__

• int apply_undo ()
  Apply the current undo/redo operation.
• void drawtext (int, int, int, int)
  Draws the text in the passed bounding box.
• void drawtext (int, int, int, int, bool draw_active)
  Draws the text in the passed bounding box.
• void handle_mouse (int, int, int, int, int keepmark=0)
  Handles mouse clicks and mouse moves.
• int handletext (int e, int, int, int)
  Handles all kinds of text field related events.
• int line_end (int i) const
  Finds the end of a line.
• int line_start (int i) const
Finds the start of a line.

- int linesPerPage ()
- void maybe_do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
- int up_down_position (int, int keepmark=0)

  Moves the cursor to the column given by up_down_pos.

- int word_end (int i) const

  Finds the end of a word.

- int word_start (int i) const

  Finds the start of a word.

- int xscroll () const
- int yscroll () const
- void yscroll (int yOffset)

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)

  Clears a flag in the flags mask.

- void draw_backdrop () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void draw_box () const

  Draws the widget box according its box style.

- void draw_box (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.

- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.

- void draw_focus () const

  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int X, int Y , int W, int H) const

  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.

- void draw_label () const

  Draws the widget's label at the defined label position.

- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)

  Creates a widget at the given position and size.

- unsigned int flags () const

  Gets the widget flags mask.

- void h (int v)

  Internal use only.

- void set_flag (unsigned int c)

  Sets a flag in the flags mask.

- void w (int v)

  Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.
Friends

- class Fl_Cocoa_Screen_Driver
- class Fl_Screen_Driver

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)
  
  The default callback for all widgets that don't set a callback.
- static unsigned int labelShortcut (const char ∗t)
  
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int testShortcut (const char ∗t, const bool require_alt=false)
  
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCEPTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }

flags possible values enumeration.

12.74.1 Detailed Description

This is the FLTK text input widget. It displays a single line of text and lets the user edit it. Normally it is drawn with an inset box and a white background. The text may contain any characters, and will correctly display any UTF text, using ^X notation for unprintable control characters. It assumes the font can draw any characters of the used scripts, which is true for standard fonts under Windows and Mac OS X. Characters can be input using the keyboard or the character palette/map. Character composition is done using dead keys and/or a compose key as defined by the operating system.

<table>
<thead>
<tr>
<th>Mouse button 1</th>
<th>Moves the cursor to this point. Drag selects characters. Double click selects words. Triple click selects all line. Shift+click extends the selection. When you select text it is automatically copied to the selection buffer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mouse button 2</td>
<td>Insert the selection buffer at the point clicked. You can also select a region and replace it with the selection buffer by selecting the region with mouse button 2.</td>
</tr>
<tr>
<td>Mouse button 3</td>
<td>Currently acts like button 1.</td>
</tr>
<tr>
<td>Backspace</td>
<td>Deletes one character to the left, or deletes the selected region.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes one character to the right, or deletes the selected region. Combine with Shift for equivalent of ^X (copy+cut).</td>
</tr>
<tr>
<td>Enter</td>
<td>May cause the callback, see when().</td>
</tr>
</tbody>
</table>

Table 12.296 Keyboard and mouse bindings.

Table 12.297 Platform specific keyboard bindings.
<table>
<thead>
<tr>
<th>Windows/Linux</th>
<th>Mac</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>^A</td>
<td>Command-A</td>
<td>Selects all text in the widget.</td>
</tr>
<tr>
<td>^C</td>
<td>Command-C</td>
<td>Copy the current selection to the clipboard.</td>
</tr>
<tr>
<td>^I</td>
<td>^I</td>
<td>Insert a tab.</td>
</tr>
</tbody>
</table>
| ^J            | ^J           | Insert a Line Feed.  
(Similar to literal 'Enter' character) |
| ^L            | ^L           | Insert a Form Feed.                             |
| ^M            | ^M           | Insert a Carriage Return.                       |
| ^V, Shift-Insert | Command-V    | Paste the clipboard.  
(Macs keyboards don't have "Insert" keys, but if they did, Shift-Insert would work) |
| ^X, Shift-Delete | Command-X, Shift-Delete | Cut.  
Copy the selection to the clipboard and delete it. (If there's no selection, Shift-Delete acts like Delete) |
| ^Z            | Command-Z    | Undo.                                            
This is a single-level undo mechanism, but all adjacent deletions and insertions are concatenated into a single "undo". Often this will undo a lot more than you expected. |
| Shift-^Z      | Shift-Command-Z | Redo.  
Currently same behavior as ^Z. Reserved for future multilevel undo/redo. |
| Arrow Keys    | Arrow Keys   | Standard cursor movement.                      
Can be combined with Shift to extend selection. |
| Home          | Command-Up, Command-Left | Move to start of line.  
Can be combined with Shift to extend selection. |
| End           | Command-Down, Command-Right | Move to end of line.  
Can be combined with Shift to extend selection. |
| Ctrl-Home     | Command-Up, Command-PgUp, Ctrl-Left | Move to top of document/field.  
In single line input, moves to start of line. In multiline input, moves to start of top line. Can be combined with Shift to extend selection. |
| Ctrl-End      | Command-End, Command-PgDn, Ctrl-Right | Move to bottom of document/field.  
In single line input, moves to end of line. In multiline input, moves to end of last line. Can be combined with Shift to extend selection. |
| Ctrl-Left     | Alt-Left     | Word left.                                      
Can be combined with Shift to extend selection. |
| Ctrl-Right    | Alt-Right    | Word right.                                     
Can be combined with Shift to extend selection. |
| Ctrl-Backspace | Alt-Backspace | Delete word left.                            |
| Ctrl-Delete   | Alt-Delete   | Delete word right.                            |
12.74 Fl_Input Class Reference

12.74.2 Constructor & Destructor Documentation

12.74.2.1 Fl_Input()

Fl_Input::Fl_Input ( 
    int X,
    int Y,
    int W,
    int H,
    const char * l = 0
) Creates a new Fl_Input widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX.

12.74.3 Member Function Documentation

12.74.3.1 draw()

void Fl_Input::draw ( ) [protected], [virtual] Draws the widget. Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead. Override this function to draw your own widgets. If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()

Implements Fl_Widget.

12.74.3.2 handle()

int Fl_Input::handle ( 
    int event ) [virtual]
Handles the specified event. You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval. One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class. In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in | event | the kind of event received |

Return values

| 0 | if the event was not used or understood |
| 1 | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Widget.
Reimplemented in Fl_Spinner::Fl_Spinner_Input, and Fl_Secret_Input.
12.74.3.3 handle_key()

int Fl_Input::handle_key () [protected]
Handles a keystroke.
This protected method handles a keystroke in an Fl_Input or derived class. It handles compose key sequences and can also be used e.g. in Fl_Multiline_Input, Fl_Float_Input and several more derived classes. The details are way too complicated to be documented here and can be changed as required. If in doubt, please consult the source code.

Returns

1 if the keystroke is handled by us, 0 if not.

12.74.3.4 handle_rmb()

int Fl_Input::handle_rmb () [protected]
Handle right mouse button down events.

Returns

1

The documentation for this class was generated from the following files:

• Fl_Input.H
• Fl_Input.cxx

12.75 Fl_Input_ Class Reference

This class provides a low-overhead text input field.
#include <Fl_Input_.H>

Inheritance diagram for Fl_Input_:

```
Fl_Widget
|    |
|    | Fl_Input
|    |
| Fl_File_Input Fl_Float_Input Fl_Int_Input Fl_Multiline_Input Fl_Output Fl_Secret_Input Fl_Spinner::Fl_Spinner_Input Fl_Multiline_Output
```

Public Member Functions

• int append (const char *t, int l=0, char keep_selection=0)
  Append text at the end.
• bool can_redo () const
  Check if there is a redo action available.
• bool can_undo () const
  Check if the last operation can be undone.
• int copy (int clipboard)
  Put the current selection into the clipboard.
• int copy_cuts ()
  Copies the yank buffer to the clipboard.
• Fl_Color cursor_color () const
  Gets the color of the cursor.
• void cursor_color (Fl_Color n)
  Sets the color of the cursor.
• int cut ()
Deletes the current selection.
• int cut (int a, int b)
  Deletes all characters between index a and b.
• int cut (int n)
  Deletes the next n bytes rounded to characters before or after the cursor.
• double dvalue () const
  Returns the widget text interpreted as a floating point number.
• Fl_Input_ (int, int, int, int, const char ∗=0)
  Creates a new Fl_Input_ widget.
• unsigned int index (int i) const
  Returns the character at index i.
• int input_type () const
  Gets the input field type.
• void input_type (int t)
  Sets the input field type.
• int insert (const char ∗t, int l=0)
  Inserts text at the cursor position.
• int insert_position () const
  Gets the position of the text cursor.
• int insert_position (int p)
  Sets the cursor position and mark.
• int insert_position (int p, int m)
  Sets the index for the cursor and mark.
• int ivalue () const
  Returns the widget text interpreted as a signed integer.
• int mark () const
  Gets the current selection mark.
• int mark (int m)
  Sets the current selection mark.
• int maximum_size () const
  Gets the maximum length of the input field in characters.
• void maximum_size (int m)
  Sets the maximum length of the input field in characters.
• int position () const
• int position (int p)
• int position (int p, int m)
• int readonly () const
  Gets the read-only state of the input field.
• void readonly (int b)
  Sets the read-only state of the input field.
• int redo ()
  Redo previous undo operation.
• int replace (int b, int e, const char ∗text, int ilen=0)
  Deletes text from b to e and inserts the new string text.
• void resize (int, int, int, int) FL_OVERRIDE
  Changes the size of the widget.
• int shortcut () const
  Return the shortcut key associated with this widget.
• void shortcut (int s)
  Sets the shortcut key associated with this widget.
• int size () const
Returns the number of bytes in value().

- void size (int W, int H)
  Sets the width and height of this widget.
- int static_value (const char * )
  Changes the widget text.
- int static_value (const char *, int)
  Changes the widget text.
- int tab_nav () const
  Gets whether the Tab key causes focus navigation in multiline input fields or not.
- void tab_nav (int val)
  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- Fl_Color textcolor () const
  Gets the color of the text in the input field.
- void textcolor (Fl_Color n)
  Sets the color of the text in the input field.
- Fl_Font textfont () const
  Gets the font of the text in the input field.
- void textfont (Fl_Font s)
  Sets the font of the text in the input field.
- Fl_Fontsize textsize () const
  Gets the size of the text in the input field.
- void textsize (Fl_Fontsize s)
  Sets the size of the text in the input field.
- int undo ()
  Undoes previous changes to the text buffer.
- const char * value () const
  Returns the text displayed in the widget.
- int value (const char *)
  Changes the widget text.
- int value (const char *, int)
  Changes the widget text.
- int value (double value)
  Changes the widget text to a floating point number ("%g").
- int value (int value)
  Changes the widget text to a signed integer number.
- int wrap () const
  Gets the word wrapping state of the input field.
- void wrap (int b)
  Sets the word wrapping state of the input field.
- ~Fl_Input_ ()
  Destroys the widget.

Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
Returns whether the widget and all of its parents are active.
• Fl_Align align () const
  Gets the label alignment.
• void align (Fl_Align alignment)
  Sets the label alignment.
• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
• virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const ∗ as_gl_window () const
• virtual Fl_Group ∗ as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• virtual Fl_Group const ∗ as_group () const
• virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const ∗ as_window () const
• void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• Fl_Boxtype box () const
  Gets the box type of the widget.
• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
• Fl_Callback_p callback () const
  Gets the current callback function for the widget.
• void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
  Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.

- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

- FL_Color color () const
  Gets the background color of the widget.

- void color (FL_Color bg)
  Sets the background color of the widget.

- void color (FL_Color bg, FL_Color sel)
  Sets the background and selection color of the widget.

- FL_Color color2 () const
  For back compatibility only.

- void color2 (unsigned a)
  For back compatibility only.

- int contains (const Fl_Widget ∗w) const
  Checks if w is a child of this widget.

- void copy_label (const char ∗new_label)
  Sets the current label.

- void copy_tooltip (const char ∗text)
  Sets the current tooltip text.

- uchar damage () const
  Returns non-zero if draw() needs to be called.

- void damage (uchar c)
  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- int damage_resize (int, int, int, int)
  Internal use only.

- void deactivate ()
  Deactivates the widget.

- FL_Image ∗deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

- const FL_Image ∗deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

- void deimage (FL_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

- void deimage (FL_Image ∗img)
  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

- void do_callback (FL_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

- void do_callback (Fl_Widget ∗widget, long arg, FL_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- void do_callback (Fl_Widget ∗widget, void ∗arg=0, FL_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- virtual void draw ()=0
  Draws the widget.

- void draw_label (int, int, int, FL_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.

• virtual int handle (int event)
  Handles the specified event.

• virtual void hide ()
  Makes a widget invisible.

• Fl_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.

• const Fl_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.

• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.

• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.

• int image_bound () const
  Returns whether the image is managed by the widget.

• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.

• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().

• const char * label () const
  Gets the current label text.

• void label (const char *text)
  Sets the current label pointer.

• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.

• Fl_Color labelcolor () const
  Gets the label color.

• void labelcolor (Fl_Color c)
  Sets the label color.

• Fl_Font labelfont () const
  Gets the font to use.

• void labelfont (Fl_Font f)
  Sets the font to use.

• Fl_Fontsize labelsize () const
  Gets the font size in pixels.

• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.

• Fl_Labeltype labeltype () const
  Gets the label type.

• void labeltype (Fl_Labeltype a)
  Sets the label type.

• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.

• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.

• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.

• unsigned int output () const
  Returns if a widget is used for output only.

• Fl_Group * parent () const
Returns a pointer to the parent widget.

• void parent (Fl_Group *p)
  Internal use only - "for hacks only".

• void position (int X, int Y)
  Repositions the window or widget.

• void redraw ()
  Schedules the drawing of the widget.

• void redraw_label ()
  Schedules the drawing of the label.

• Fl_Color selection_color () const
  Gets the selection color.

• void selection_color (Fl_Color a)
  Sets the selection color.

• void set_active ()
  Marks the widget as active without sending events or changing focus.

• void set_changed ()
  Marks the value of the widget as changed.

• void set_output ()
  Sets a widget to output only.

• void set_visible ()
  Makes the widget visible.

• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.

• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.

• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

• virtual void show ()
  Makes a widget visible.

• void size (int W, int H)
  Changes the size of the widget.

• int take_focus ()
  Gives the widget the keyboard focus.

• unsigned int takesevents () const
  Returns if the widget is able to take events.

• int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.

• const char * tooltip () const
  Gets the current tooltip text.

• void tooltip (const char *text)
  Sets the current tooltip text.

• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const
  Gets the widget type.

• void type (uchar t)
  Sets the widget type.

• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void *\texttt{user\_data} () const
  
  \textit{Gets the user data for this widget.}

• void \texttt{user\_data} (\texttt{Fl\_Callback\_User\_Data} *v, bool auto\_free)
  
  \textit{Sets the user data for this widget.}

• void \texttt{user\_data} (void *v)
  
  \textit{Sets the user data for this widget.}

• unsigned int \texttt{visible} () const
  
  \textit{Returns whether a widget is visible.}

• unsigned int \texttt{visible\_focus} () const
  
  \textit{Checks whether this widget has a visible focus.}

• void \texttt{visible\_focus} (int v)
  
  \textit{Modifies keyboard focus navigation.}

• int \texttt{visible\_r} () const
  
  \textit{Returns whether a widget and all its parents are visible.}

• int \texttt{w} () const
  
  \textit{Gets the widget width.}

• \texttt{Fl\_When} \texttt{when} () const
  
  \textit{Returns the conditions under which the callback is called.}

• void \texttt{when} (uchar i)
  
  \textit{Sets the flags used to decide when a callback is called.}

• \texttt{Fl\_Window} *\texttt{window} () const
  
  \textit{Returns a pointer to the nearest parent window up the widget hierarchy.}

• int \texttt{x} () const
  
  \textit{Gets the widget position in its window.}

• int \texttt{y} () const
  
  \textit{Gets the widget position in its window.}

• virtual ~\texttt{Fl\_Widget} ()
  
  \textit{Destroys the widget.}

\begin{description}
\item[Protected Member Functions]
• int \texttt{apply\_undo} ()
  
  \textit{Apply the current undo\/redo operation.}

• void \texttt{drawtext} (int, int, int, int)
  
  \textit{Draws the text in the passed bounding box.}

• void \texttt{drawtext} (int, int, int, bool draw\_active)
  
  \textit{Draws the text in the passed bounding box.}

• void \texttt{handle\_mouse} (int, int, int, int keepmark=0)
  
  \textit{Handles mouse clicks and mouse moves.}

• int \texttt{handle\_text} (int e, int, int, int)
  
  \textit{Handles all kinds of text field related events.}

• int \texttt{line\_end} (int i) const
  
  \textit{Finds the end of a line.}

• int \texttt{line\_start} (int i) const
  
  \textit{Finds the start of a line.}

• int \texttt{lines\_per\_page} ()
  
  \textit{Moves the cursor to the column given by \texttt{up\_down\_pos}.}

• int \texttt{word\_end} (int i) const
  
  \textit{Finds the end of a word.}
\end{description}
• int word_start (int i) const
  Finds the start of a word.
• int xscroll () const
• int yscroll () const
• void yscroll (int yOffset)

Protected Member Functions inherited from Fl_Widget
• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const
  Draws the widget box according its box style.
• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
• void draw_focus () const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
• void draw_label () const
  Draws the widget's label at the defined label position.
• void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
• Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.
• unsigned int flags () const
  Gets the widget flags mask.
• void h (int v)
  Internal use only.
• void set_flag (unsigned int c)
  Sets a flag in the flags mask.
• void w (int v)
  Internal use only.
• void x (int v)
  Internal use only.
• void y (int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget
• static void default_callback (Fl_Widget ∗widget, void ∗data)
  The default callback for all widgets that don't set a callback.
• static unsigned int label_shortcut (const char ∗t)
  Returns the Unicode value of the '&x' shortcut in a given text.
• static int test_shortcut (const char ∗, const bool require_alt=false)
  Returns true if the given text ∗t contains the entered '&x' shortcut.
### Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1 << 0, INVISIBLE = 1 << 1, OUTPUT = 1 << 2, NOBORDER = 1 << 3,
  FORCE_POSITION = 1 << 4, NON_MODAL = 1 << 5, SHORTCUT_LABEL = 1 << 6, CHANGED = 1 << 7,
  OVERRIDE = 1 << 8, VISIBLE_FOCUS = 1 << 9, COPIED_LABEL = 1 << 10, CLIP_CHILDREN = 1 << 11,
  MENU_WINDOW = 1 << 12, TOOLTIP_WINDOW = 1 << 13, MODAL = 1 << 14, NO_OVERLAY = 1 << 15,
  GROUP_RELATIVE = 1 << 16, COPIED_TOOLTIP = 1 << 17, FULLSCREEN = 1 << 18, MAC_USE_ACCENTS_MENU = 1 << 19,
  NEEDS_KEYBOARD = 1 << 20, IMAGE_BOUND = 1 << 21, DEIMAGE_BOUND = 1 << 22, AUTO_DELETE_USER_DATA = 1 << 23,
  MAXIMIZED = 1 << 24, USERFLAG3 = 1 << 29, USERFLAG2 = 1 << 30, USERFLAG1 = 1 << 31
}

*flags possible values enumeration.*

### 12.75.1 Detailed Description

This class provides a low-overhead text input field. This is a virtual base class below Fl_Input. It has all the same interfaces, but lacks the handle() and draw() method. You may want to subclass it if you are one of those people who likes to change how the editing keys work. It may also be useful for adding scrollbars to the input field.

This can act like any of the subclasses of Fl_Input, by setting type() to one of the following values:

- `#define FL_NORMAL_INPUT 0`
- `#define FL_FLOAT_INPUT 1`
- `#define FL_INT_INPUT 2`
- `#define FL_MULTILINE_INPUT 4`
- `#define FL_SECRET_INPUT 5`
- `#define FL_INPUT_TYPE 7`
- `#define FL_INPUT_READONLY 8`
- `#define FL_NORMAL_OUTPUT (FL_NORMAL_INPUT | FL_INPUT_READONLY)`
- `#define FL_MULTILINE_OUTPUT (FL_MULTILINE_INPUT | FL_INPUT_READONLY)`
- `#define FL_INPUT_WRAP 16`
- `#define FL_MULTILINE_INPUT_WRAP (FL_MULTILINE_INPUT | FL_INPUT_READONLY | FL_INPUT_WRAP)`
- `#define FL_MULTILINE_OUTPUT_WRAP (FL_MULTILINE_INPUT | FL_INPUT_READONLY | FL_INPUT_WRAP)`

All variables that represent an index into a text buffer are byte-oriented, not character oriented, counting from 0 (at or before the first character) to size() (at the end of the buffer, after the last byte). Since UTF-8 characters can be up to six bytes long, simply incrementing such an index will not reliably advance to the next character in the text buffer. Indices and pointers into the text buffer should always point at a 7 bit ASCII character or the beginning of a UTF-8 character sequence. Behavior for false UTF-8 sequences and pointers into the middle of a sequence are undefined.

See also

- Fl_Text_Display, Fl_Text_Editor for more powerful text handling widgets
- Fl_Widget::shortcut_label(int)

### 12.75.2 Constructor & Destructor Documentation

#### 12.75.2.1 Fl_Input()

`Fl_Input_::Fl_Input_ (int X, int Y, int W, int H, const char * l = 0)`

Creates a new Fl_Input_ widget.

This function creates a new Fl_Input_ widget and adds it to the current Fl_Group. The value() is set to NULL. The default boxtype is FL_DOWN_BOX.

**Parameters**

- **X, Y, W, H**: the dimensions of the new widget
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{l}</td>
<td>an optional label text</td>
</tr>
</tbody>
</table>

12.75.2.2 \texttt{\textasciitilde Fl\_Input\_()} \par
\texttt{Fl\_Input\_::\textasciitilde Fl\_Input\_ ( )}

Destroys the widget.
The destructor clears all allocated buffers and removes the widget from the parent \texttt{Fl\_Group}.

12.75.3 Member Function Documentation

12.75.3.1 append() \par
\texttt{int Fl\_Input\_::append ( const char * \textit{t}, int \textit{l} = 0, char keep\_selection = 0 )}

Append text at the end.
This function appends the string in \textit{t} to the end of the text. It does not moves the new position or mark.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{t}</td>
<td>text that will be appended</td>
</tr>
<tr>
<td>\textit{l}</td>
<td>length of text, or 0 if the string is terminated by nul.</td>
</tr>
<tr>
<td>\textit{keep_selection}</td>
<td>if this is 1, the current text selection will remain, if 0, the cursor will move to the end of the inserted text.</td>
</tr>
</tbody>
</table>

Returns

0 if no text was appended

12.75.3.2 apply\_undo() \par
\texttt{int Fl\_Input\_::apply\_undo ( ) [protected]}

Apply the current undo/redo operation.
It's up to \texttt{undo()} and \texttt{redo()} to push and pop actions to and from the lists.

Returns

1 if the current action changed any text.

See also

\texttt{undo()}, \texttt{redo()}

12.75.3.3 can\_redo() \par
\texttt{bool Fl\_Input\_::can\_redo ( ) const}

Check if there is a redo action available.

Returns

true if the widget can redo the last undo action
12.75.3.4 can_undo()

bool Fl_Input_::can_undo () const
Check if the last operation can be undone.

Returns
true if the widget can undo the last change

12.75.3.5 copy()

int Fl_Input_::copy ( int clipboard )
Put the current selection into the clipboard.
This function copies the current selection between mark() and position() into the specified clipboard. This does not replace the old clipboard contents if position() and mark() are equal. Clipboard 0 maps to the current text selection and clipboard 1 maps to the cut/paste clipboard.

Parameters

| clipboard | the clipboard destination 0 or 1 |

Returns
0 if no text is selected, 1 if the selection was copied

See also
Fl::copy(const char *, int, int)

12.75.3.6 copy_cuts()

int Fl_Input_::copy_cuts ( )
Copies the yank buffer to the clipboard.
This method copies all the previous contiguous cuts from the undo information to the clipboard. This function implements the ^K shortcut key.

Returns
0 if the operation did not change the clipboard

See also
copy(int), cut

12.75.3.7 cursor_color() [1/2]

Fl_Color Fl_Input_::cursor_color ( ) const [inline]
Gets the color of the cursor.

Returns
the current cursor color

12.75.3.8 cursor_color() [2/2]

void Fl_Input_::cursor_color ( Fl_Color n ) [inline]
Sets the color of the cursor.
The default color for the cursor is FL_BLACK.
Parameters

| in  | n | the new cursor color |

12.75.3.9  **cut()** [1/3]

```cpp
int Fl_Input_::cut ( ) [inline]
```

Deletes the current selection.
This function deletes the currently selected text *without* storing it in the clipboard. To use the clipboard, you may call `copy()` first or `copy_cuts()` after this call.

Returns

0 if no data was copied

12.75.3.10  **cut()** [2/3]

```cpp
int Fl_Input_::cut ( int a, int b ) [inline]
```

Deletes all characters between index a and b.
This function deletes the currently selected text *without* storing it in the clipboard. To use the clipboard, you may call `copy()` first or `copy_cuts()` after this call.

Parameters

| a,b | range of bytes rounded to full characters and clamped to the buffer |

Returns

0 if no data was copied

12.75.3.11  **cut()** [3/3]

```cpp
int Fl_Input_::cut ( int n ) [inline]
```

Deletes the next n bytes rounded to characters before or after the cursor.
This function deletes the currently selected text *without* storing it in the clipboard. To use the clipboard, you may call `copy()` first or `copy_cuts()` after this call.

Parameters

| n | number of bytes rounded to full characters and clamped to the buffer. A negative number will cut characters to the left of the cursor. |

Returns

0 if no data was copied

12.75.3.12  **drawtext()** [1/2]

```cpp
void Fl_Input_::drawtext ( int X, int Y, int W, int H ) [protected]
```

Generated by Doxygen
Draws the text in the passed bounding box. If `damage() & FL_DAMAGE_ALL` is true, this assumes the area has already been erased to `color()`. Otherwise it does minimal update and erases the area itself.

Parameters

| X, Y, W, H | area that must be redrawn |

12.75.3.13 `drawtext()` [2/2]

```cpp
void Fl_Input_::drawtext (int X, int Y, int W, int H, bool draw_active) [protected]
```

Draws the text in the passed bounding box. This version of `drawtext` allows the user to control whether the widget is drawn as active, i.e. with the text cursor, or inactive. This is useful for compound widgets where the input should be shown as active when actually the container widget is the active one.

A caller should not draw the widget with active set if another text widget may indeed be the active widget.

Parameters

| X, Y, W, H | area that must be redrawn |
| draw_active | if set, the cursor will be drawn, even if the widget is not active |

See also

`Fl_Input_::drawtext(int X, int Y, int W, int H)`

12.75.3.14 `dvalue()`

```cpp
double Fl_Input_::dvalue ( ) const
```

Returns the widget text interpreted as a floating point number.

Returns

double precision floating point value

See also

`Fl_Input_::ivalue()`
`Fl_Input_::value(double)`

12.75.3.15 `handle_mouse()`

```cpp
void Fl_Input_::handle_mouse (int X, int Y, int , int , int drag = 0) [protected]
```

Handles mouse clicks and mouse moves.

Todo Add comment and parameters
12.75.3.16 handletext()

```cpp
int Fl_Input_::handletext (  
    int event,  
    int X,  
    int Y,  
    int W,  
    int H ) [protected]
```

Handles all kinds of text field related events. This is called by derived classes.

**Todo** Add comment and parameters

12.75.3.17 index()

```cpp
unsigned int Fl_Input_::index (  
    int i ) const
```

Returns the character at index \( i \). This function returns the UTF-8 character at \( i \) as a ucs4 character code.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>index into the value field</th>
</tr>
</thead>
</table>

Returns

the character at index \( i \)

12.75.3.18 input_type() [1/2]

```cpp
int Fl_Input_::input_type ( ) const [inline]
```

Gets the input field type.

Returns

the current input type

12.75.3.19 input_type() [2/2]

```cpp
void Fl_Input_::input_type (  
    int t ) [inline]
```

Sets the input field type. A **redraw()** is required to reformat the input field.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>new input type</th>
</tr>
</thead>
</table>

12.75.3.20 insert()

```cpp
int Fl_Input_::insert (  
    const char * t,  
    int l = 0 ) [inline]
```

Inserts text at the cursor position. This function inserts the string in \( t \) at the cursor **position()** and moves the new position and mark to the end of the inserted text.
Parameters

<table>
<thead>
<tr>
<th>in  t</th>
<th>text that will be inserted</th>
</tr>
</thead>
<tbody>
<tr>
<td>in   l</td>
<td>length of text, or 0 if the string is terminated by nul.</td>
</tr>
</tbody>
</table>

Returns

0 if no text was inserted

12.75.3.21 insert_position() [1/3]

```cpp
int Fl_Input_::insert_position ( ) const [inline]
```

Gets the position of the text cursor.

**Returns**

the cursor position as an index in the range 0..size()

**See also**

insert_position(int, int)

12.75.3.22 insert_position() [2/3]

```cpp
int Fl_Input_::insert_position ( int p ) [inline]
```

Sets the cursor position and mark. position(n) is the same as position(n, n).

**Parameters**

| p   | new index for cursor and mark |

**Returns**

0 if no positions changed

**See also**

insert_position(int, int), insert_position(), mark(int)

12.75.3.23 insert_position() [3/3]

```cpp
int Fl_Input_::insert_position ( int p, int m )
```

Sets the index for the cursor and mark. The input widget maintains two pointers into the string. The position (p) is where the cursor is. The mark (m) is the other end of the selected text. If they are equal then there is no selection. Changing this does not affect the clipboard (use copy() to do that). Changing these values causes a redraw(). The new values are bounds checked.

**Parameters**

| p | index for the cursor position |
| m | index for the mark |
Returns
0 if no positions changed

See also
position(int), position(), mark(int)

12.75.3.24  ivalue()

int Fl_Input_::ivalue ( ) const
Returns the widget text interpreted as a signed integer.

Returns
signed integer value

See also
Fl_Input_::dvalue()
Fl_Input_::value(int)

12.75.3.25  line_end()

int Fl_Input_::line_end ( int i ) const [protected]
Finds the end of a line.
This call calculates the end of a line based on the given index i.

Parameters

| in | i | starting index for the search |

Returns
dead of the line

12.75.3.26  line_start()

int Fl_Input_::line_start ( int i ) const [protected]
Finds the start of a line.
This call calculates the start of a line based on the given index i.

Parameters

| in | i | starting index for the search |

Returns
start of the line

12.75.3.27  mark() [1/2]

int Fl_Input_::mark ( ) const [inline]
Gets the current selection mark.
12.75.3.28 \texttt{mark()} [2/2]

\begin{verbatim}
int Fl_Input_::mark (  
  int m ) [inline]
\end{verbatim}

Sets the current selection mark. \texttt{mark(n)} is the same as \texttt{insert_position(insert_position(),n)}.

Parameters

| \texttt{m} | new index of the mark |

Returns

0 if the mark did not change

See also

\texttt{insert_position()}, \texttt{insert_position(int, int)}

12.75.3.29 \texttt{maximum_size()} [1/2]

\begin{verbatim}
int Fl_Input_::maximum_size ( ) const [inline]
\end{verbatim}

Gets the maximum length of the input field in characters.

See also

\texttt{maximum_size(int)}.

12.75.3.30 \texttt{maximum_size()} [2/2]

\begin{verbatim}
void Fl_Input_::maximum_size (  
  int m ) [inline]
\end{verbatim}

Sets the maximum length of the input field in characters. This limits the number of characters that can be inserted in the widget. Since FLTK 1.3 this is different than the buffer size, since one character can be more than one byte in UTF-8 encoding. In FLTK 1.1 this was the same (one byte = one character).

12.75.3.31 \texttt{position()} [1/3]

\begin{verbatim}
int Fl_Input_::position ( ) const [inline]
\end{verbatim}

\textbf{Deprecated} “in 1.4.0 - use insert_position() instead”

12.75.3.32 \texttt{position()} [2/3]

\begin{verbatim}
int Fl_Input_::position (  
  int p ) [inline]
\end{verbatim}

\textbf{Deprecated} “in 1.4.0 - use insert_position(p) instead”
12.75.3.33  position() [3/3]

int Fl_Input_::position (  
    int p,  
    int m ) [inline]

Deprecated  "in 1.4.0 - use insert_position(p, m) or Fl_Widget::position(x, y) instead"

12.75.3.34  readonly() [1/2]

int Fl_Input_::readonly ( ) const [inline]

Gets the read-only state of the input field.

Returns

non-zero if this widget is read-only

12.75.3.35  readonly() [2/2]

void Fl_Input_::readonly (  
    int b ) [inline]

Sets the read-only state of the input field.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>if b is 0, the text in this widget can be edited by the user</td>
<td></td>
</tr>
</tbody>
</table>

12.75.3.36  redo()

int Fl_Input_::redo ( )

Redo previous undo operation.

This call reappplies previously executed undo operations.

Returns

non-zero if any change was made.

12.75.3.37  replace()

int Fl_Input_::replace (  
    int b,  
    int e,  
    const char * text,  
    int ilen = 0 )

Deletes text from b to e and inserts the new string text.

All changes to the text buffer go through this function. It deletes the region between b and e (either one may be less or equal to the other), and then inserts the string text at that point and moves the mark() and position() to the end of the insertion. Does the callback if when() & FL_WHEN_CHANGED and there is a change.

Set b and e equal to not delete anything. Set text to NULL to not insert anything.

ilen can be zero or strlen(text), which saves a tiny bit of time if you happen to already know the length of the insertion, or can be used to insert a portion of a string. If ilen is zero, strlen(text) is used instead. b and e are clamped to the 0..size() range, so it is safe to pass any values. b, e, and ilen are used as numbers of bytes (not characters), where b and e count from 0 to size() (end of buffer).

If b and/or e don't point to a valid UTF-8 character boundary, they are adjusted to the previous (b) or the next (e) valid UTF-8 character boundary, resp..

If the current number of characters in the buffer minus deleted characters plus inserted characters in text would overflow the number of allowed characters (maximum_size()), then only the first characters of the string are inserted, so that maximum_size() is not exceeded.
cut() and insert() are just inline functions that call replace().

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in b</code></td>
<td>beginning index of text to be deleted</td>
</tr>
<tr>
<td><code>in e</code></td>
<td>ending index of text to be deleted and insertion position</td>
</tr>
<tr>
<td><code>in text</code></td>
<td>string that will be inserted</td>
</tr>
<tr>
<td><code>in ilen</code></td>
<td>length of text or 0 for null terminated strings</td>
</tr>
</tbody>
</table>

Returns
0 if nothing changed

Note
If `text` does not point to a valid UTF-8 character or includes invalid UTF-8 sequences, the text is inserted nevertheless (counting invalid UTF-8 bytes as one character each).

12.75.3.38 resize()

```cpp
void Fl_Input_::resize ( int X, int Y, int W, int H ) [virtual]
```

Changes the size of the widget.
This call updates the text layout so that the cursor is visible.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in X,Y,W,H</code></td>
<td>new size of the widget</td>
</tr>
</tbody>
</table>

See also
Fl_Widget::resize(int, int, int)
Reimplemented from Fl_Widget.

12.75.3.39 shortcut() [1/2]

```cpp
int Fl_Input_::shortcut ( ) const [inline]
```

Return the shortcut key associated with this widget.

Returns
shortcut keystroke

See also
Fl_Button::shortcut()

12.75.3.40 shortcut() [2/2]

```cpp
void Fl_Input_::shortcut ( int s ) [inline]
```

Sets the shortcut key associated with this widget.
Pressing the shortcut key gives text editing focus to this widget.
Parameters

| in | s  | new shortcut keystroke |

See also

Fl_Button::shortcut()

12.75.3.41 size() [1/2]

int Fl_Input_::size () const [inline]

Returns the number of bytes in value().
This may be greater than strlen(value()) if there are null characters in the text.

Returns

number of bytes in the text

12.75.3.42 size() [2/2]

void Fl_Input_::size (  
  int W, 
  int H ) [inline]

Sets the width and height of this widget.

Parameters

| in | W, H | new width and height |

See also

Fl_Widget::size(int, int)

12.75.3.43 static_value() [1/2]

int Fl_Input_::static_value (  
  const char * str )

Changes the widget text.
This function changes the text and sets the mark and the point to the end of it. The string is not copied. If the user edits the string it is copied to the internal buffer then. This can save a great deal of time and memory if your program is rapidly changing the values of text fields, but this will only work if the passed string remains unchanged until either the Fl_Input is destroyed or value() is called again.

Parameters

| in | str | the new text |

Returns

non-zero if the new value is different than the current one

12.75.3.44 static_value() [2/2]

int Fl_Input_::static_value (  
  const char * str, 
  int len )
Changes the widget text. This function changes the text and sets the mark and the point to the end of it. The string is not copied. If the user edits the string it is copied to the internal buffer then. This can save a great deal of time and memory if your program is rapidly changing the values of text fields, but this will only work if the passed string remains unchanged until either the Fl_Input is destroyed or value() is called again. You can use the len parameter to directly set the length if you know it already or want to put null characters in the text.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>str</th>
<th>the new text</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>len</td>
<td>the length of the new text</td>
</tr>
</tbody>
</table>

Returns

non-zero if the new value is different than the current one

12.75.3.45 tab_nav() [1/2]

int Fl_Input_::tab_nav ( ) const [inline]
GETS WHETHER THE TAB KEY CAUSES FOCUS NAVIGATION IN MULTILINE INPUT FIELDS OR NOT. IF ENABLED (DEFAULT), HITTING TAB CAUSES FOCUS NAVIGATION TO THE NEXT Widget. IF DISABLED, HITTING TAB INSERTS A TAB CHARACTER INTO THE TEXT FIELD.

Returns

1 if Tab advances focus (default), 0 if Tab inserts tab characters.

See also

tab_nav(int), Fl::OPTION_ARROW_FOCUS.

12.75.3.46 tab_nav() [2/2]

void Fl_Input_::tab_nav ( int val ) [inline]

SETS WHETHER THE TAB KEY DOES FOCUS NAVIGATION, OR INSERTS TAB CHARACTERS INTO Fl_Multiline_Input. By default this flag is enabled to provide the 'normal' behavior most users expect; Tab navigates focus to the next widget. To inserting an actual Tab character, users can use Ctrl-I or copy/paste. Disabling this flag gives the old FLTK behavior where Tab inserts a tab character into the text field, in which case only the mouse can be used to navigate to the next field. History: This flag was provided for backwards support of FLTK's old 1.1.x behavior where Tab inserts a tab character instead of navigating focus to the next widget. This behavior was unique to Fl_Multiline_Input. With the advent of Fl_Text_Editor, this old behavior has been deprecated.

Parameters

| in   | val | If val is 1, Tab advances focus (default). If val is 0, Tab inserts a tab character (old FLTK behavior). |

See also

tab_nav(), Fl::OPTION_ARROW_FOCUS.

12.75.3.47 textcolor() [1/2]

Fl_Color Fl_Input_::textcolor ( ) const [inline]

GETS THE COLOR OF THE TEXT IN THE INPUT FIELD.
Returns

the text color

See also

textcolor(Fl_Color)

12.75.3.48 textcolor() [2/2]

void Fl_Input_::textcolor (  
    Fl_Color n ) [inline]

Sets the color of the text in the input field.
The text color defaults to FL_FOREGROUND_COLOR.

Parameters

| in | n | new text color |

See also

textcolor()

12.75.3.49 textfont() [1/2]

Fl_Font Fl_Input_::textfont ( ) const [inline]

Gets the font of the text in the input field.

Returns

the current Fl_Font index

12.75.3.50 textfont() [2/2]

void Fl_Input_::textfont (  
    Fl_Font s ) [inline]

Sets the font of the text in the input field.
The text font defaults to FL_HELVETICA.

Parameters

| in | s | the new text font |

12.75.3.51 textsize() [1/2]

Fl_Fontsize Fl_Input_::textsize ( ) const [inline]

 Gets the size of the text in the input field.

Returns

the text height in pixels

12.75.3.52 textsize() [2/2]

void Fl_Input_::textsize (  
    Fl_Fontsize s ) [inline]

Sets the size of the text in the input field.
The text height defaults to `FL_NORMAL_SIZE`. 


Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>the new font height in pixel units</th>
</tr>
</thead>
</table>

12.75.3.53  **undo()**

```cpp
int Fl_Input_::undo ( )
```

Undoes previous changes to the text buffer.
This call undoes a number of previous calls to `replace()`.

Returns

non-zero if any change was made.

12.75.3.54  **up_down_position()**

```cpp
int Fl_Input_::up_down_position ( int i,
int keepmark = 0 ) [protected]
```

Moves the cursor to the column given by `up_down_pos`.
This function is helpful when implementing up and down cursor movement. It moves the cursor from the beginning of a line to the column indicated by the global variable `up_down_pos` in pixel units.

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>index into the beginning of a line of text</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i</td>
<td></td>
</tr>
<tr>
<td></td>
<td>keepmark</td>
<td>if set, move only the cursor, but not the mark</td>
</tr>
</tbody>
</table>

Returns

index to new cursor position

12.75.3.55  **value()** [1/5]

```cpp
const char * Fl_Input_::value ( ) const [inline]
```

Returns the text displayed in the widget.
This function returns the current value, which is a pointer to the internal buffer and is valid only until the next event is handled.

Returns

pointer to an internal buffer - do not free() this

See also

`Fl_Input_::value(const char*)`

12.75.3.56  **value()** [2/5]

```cpp
int Fl_Input_::value ( const char * str )
```

Changes the widget text.
This function changes the text and sets the mark and the point to the end of it. The string is copied to the internal buffer. Passing NULL is the same as " ".

Generated by Doxygen
Parameters

| in  | str  | the new text |

Returns

non-zero if the new value is different than the current one

See also

Fl_Input_::value(const char * str, int len), Fl_Input_::value()

### 12.75.3.57 value() [3/5]

```c
int Fl_Input_::value (const char * str,
                       int len )
```

Changes the widget text. This function changes the text and sets the mark and the point to the end of it. The string is copied to the internal buffer. Passing NULL is the same as "". You can use the length parameter to directly set the length if you know it already or want to put null characters in the text.

Parameters

| in  | str  | the new text |

| in  | len  | the length of the new text |

Returns

non-zero if the new value is different than the current one

See also

Fl_Input_::value(const char * str), Fl_Input_::value()

### 12.75.3.58 value() [4/5]

```c
int Fl_Input_::value (double v )
```

Changes the widget text to a floating point number ("%g").

Parameters

| in  | v    | the new value |

Returns

non-zero if the new value is different than the current one

See also

Fl_Input_::value(const char * str), Fl_Input_::ivalue()

### 12.75.3.59 value() [5/5]

```c
int Fl_Input_::value (int v )
```
Changes the widget text to a signed integer number.

Parameters

\begin{verbatim}
in \texttt{v}
\end{verbatim}

the new value

Returns

non-zero if the new value is different than the current one

See also

\texttt{Fl_Input::value(const char* str)}, \texttt{Fl_Input::ivalue()}

\subsection{word_end()}

\begin{verbatim}
int Fl_Input::word_end ( int \texttt{i} ) const [protected]
\end{verbatim}

Finds the end of a word.

Returns the index after the last byte of a word. If the index is already at the end of a word, it will find the end of the following word, so if you call it repeatedly you will move forwards to the end of the text.

Note that this is inconsistent with line_end().

Parameters

\begin{verbatim}
in \texttt{i} starting index for the search
\end{verbatim}

Returns

end of the word

\subsection{word_start()}

\begin{verbatim}
int Fl_Input::word_start ( int \texttt{i} ) const [protected]
\end{verbatim}

Finds the start of a word.

Returns the index of the first byte of a word. If the index is already at the beginning of a word, it will find the beginning of the previous word, so if you call it repeatedly you will move backwards to the beginning of the text.

Note that this is inconsistent with line_start().

Parameters

\begin{verbatim}
in \texttt{i} starting index for the search
\end{verbatim}

Returns

start of the word, or previous word

\subsection{wrap() [1/2]}

\begin{verbatim}
int Fl_Input::wrap ( ) const [inline]
\end{verbatim}

Gets the word wrapping state of the input field.

Word wrap is only functional with multi-line input fields.

\subsection{wrap() [2/2]}

\begin{verbatim}
void Fl_Input::wrap ( 
\end{verbatim}
int b ) [inline]
Sets the word wrapping state of the input field.
Word wrap is only functional with multi-line input fields.
The documentation for this class was generated from the following files:

- Fl_Input_.H
- Fl_Input_.cxx

## 12.76 Fl_Input_Choice Class Reference

A combination of the input widget and a menu button.

Inheritance diagram for Fl_Input_Choice:

```
Fl_Widget
   ↓
Fl_Group
   ↓
Fl_Input_Choice
```

### Public Member Functions

- **void add (const char *s)**
  
  Adds an item to the menu.

- **int changed () const**
  
  Returns the combined `changed()` state of the input and menu button widget.

- **void clear ()**
  
  Removes all items from the menu.

- **void clear_changed ()**
  
  Clears the `changed()` state of both input and menu button widgets.

- **FL_Boxtype down_box () const**
  
  Gets the box type of the menu button.

- **void down_box (FL_Boxtype b)**
  
  Sets the box type of the menu button.

- **Fl_Input_Choice (int X, int Y, int W, int H, const char *L=0)**
  
  Creates a new Fl_Input_Choice widget using the given position, size, and label string.

- **Fl_Input * input ()**
  
  Returns a pointer to the internal Fl_Input widget.

- **const Fl_Menu_Item * menu ()**
  
  Gets the Fl_Menu_Item array used for the menu.

- **void menu (const Fl_Menu_Item *m)**
  
  Sets the Fl_Menu_Item array used for the menu.

- **Fl_Menu_Button * menubutton ()**
  
  Returns a pointer to the internal Fl_Menu_Button widget.

- **void resize (int X, int Y, int W, int H) FL_OVERRIDE**
  
  Resizes the Fl_Input_Choice widget.

- **void set_changed ()**
  
  Sets the `changed()` state of both input and menu button widgets to the specified value.

- **FL_Color textcolor () const**
  
  Gets the Fl_Input text field's text color.

- **void textcolor (FL_Color c)**
Sets the Fl_Input text field's text color to \texttt{c}.

\begin{itemize}
  \item \texttt{Fl_Font textfont () const}
    \hfill Gets the Fl_Input text field's font style.
  \item \texttt{void textfont (Fl_Font f)}
    \hfill Sets the Fl_Input text field's font style to \texttt{f}.
  \item \texttt{Fl_Fontsize textsize () const}
    \hfill Gets the Fl_Input text field's font size.
  \item \texttt{void textsize (Fl_Fontsize s)}
    \hfill Sets the Fl_Input text field's font size to \texttt{s}.
  \item \texttt{int update_menubutton ()}
    \hfill Updates the menubutton with the string value in Fl_Input.
  \item \texttt{const char * value () const}
    \hfill Returns the Fl_Input text field's current contents.
  \item \texttt{void value (const char * val)}
    \hfill Sets the Fl_Input text field's contents to \texttt{val}.
  \item \texttt{void value (int val)}
    \hfill Chooses item\# \texttt{val} in the menu, and sets the Fl_Input text field to that value.
\end{itemize}

Public Member Functions inherited from Fl_Group

\begin{itemize}
  \item \texttt{Fl_Widget * & _ddfdesign_kludge ()}
    \hfill This is for forms compatibility only.
  \item \texttt{void add (Fl_Widget &)}
    \hfill The widget is removed from its current group (if any) and then added to the end of this group.
  \item \texttt{void add (Fl_Widget * o)}
    \hfill See \texttt{void Fl_Group::add(Fl_Widget &w)}
  \item \texttt{void add_resizable (Fl_Widget &o)}
    \hfill Adds a widget to the group and makes it the resizable widget.
  \item \texttt{Fl_Widget *& array () const}
    \hfill Returns a pointer to the array of children.
  \item \texttt{Fl_Group const * as_group () const FL_OVERRIDE}
    \hfill \texttt{Fl_Group} pointer if this widget is an \texttt{Fl_Group}.
  \item \texttt{void begin ()}
    \hfill Sets the current group so you can build the widget tree by just constructing the widgets.
  \item \texttt{Fl_Widget * child (int n) const}
    \hfill Returns \texttt{array()[n]}.
  \item \texttt{int children () const}
    \hfill Returns how many child widgets the group has.
  \item \texttt{void clear ()}
    \hfill Deletes all child widgets from memory recursively.
  \item \texttt{unsigned int clip_children ()}
    \hfill Returns the current clipping mode.
  \item \texttt{void clip_children (int c)}
    \hfill Controls whether the group widget clips the drawing of child widgets to its bounding box.
  \item \texttt{virtual int delete_child (int n)}
    \hfill Removes the widget at \texttt{index} from the group and deletes it.
  \item \texttt{void end ()}
    \hfill Exactly the same as \texttt{current(this->parent())}.
  \item \texttt{int find (const Fl_Widget &o) const}
    \hfill See \texttt{int Fl_Group::find(const Fl_Widget *w) const}.
\end{itemize}
• int find (const Fl_Widget *) const
  Searches the child array for the widget and returns the index.

• Fl_Group (int, int, int, const char * = 0)
  Creates a new Fl_Group widget using the given position, size, and label string.

• void focus (Fl_Widget *W)

• void forms_end ()
  This is for forms compatibility only.

• int handle (int) FL_OVERRIDE
  Handles the specified event.

• void init_sizes ()
  Resets the internal array of widget sizes and positions.

• void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.

• void insert (Fl_Widget &, Fl_Widget *before)
  This does insert(w, find(before)).

• void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.

• void remove (Fl_Widget *o)
  Removes the widget o from the group.

• void remove (int index)
  Removes the widget at index from the group but does not delete it.

• Fl_Widget * resizable () const
  Returns the group's resizable widget.

• void resizable (Fl_Widget &o)
  Sets the group's resizable widget.

• void resizable (Fl_Widget *o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

• virtual ~Fl_Group ()
  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  Activates the widget.

• unsigned int active () const
  Returns whether the widget is active.

• int active_r () const
  Returns whether the widget and all of its parents are active.

• Fl_Align align () const
  Gets the label alignment.

• void align (Fl_Align alignment)
  Sets the label alignment.

• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.

• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const * as_gl_window () const
• virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const * as_window () const
• void bind_deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• Fl_Boxtype box () const
  Gets the box type of the widget.
• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
• Fl_Callback_p callback () const
  Gets the current callback function for the widget.
• void callback (Fl_Callback *cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
  Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.
• void copy_label (const char *new_label)
  Sets the current label.
• void copy_tooltip (const char *text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image *image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image *image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
Returns whether the current label was assigned with `copy_label()`.

- `const char * label () const`
  Gets the current label text.

- `void label (const char *text)`
  Sets the current label pointer.

- `void label (Fi_Labeltype a, const char *b)`
  
  Shortcut to set the label text and type in one call.

- `Fi_Color labelcolor () const`
  Gets the label color.

- `void labelcolor (Fi_Color c)`
  Sets the label color.

- `Fi_Font labelfont () const`
  Gets the font to use.

- `void labelfont (Fi_Font f)`
  Sets the font to use.

- `Fi_Fontsize labelsize () const`
  Gets the font size in pixels.

- `void labelsize (Fi_Fontsize pix)`
  Sets the font size in pixels.

- `Fi_Labeltype labeltype () const`
  Gets the label type.

- `void labeltype (Fi_Labeltype a)`
  Sets the label type.

- `void measure_label (int &ww, int &hh) const`
  Sets width `ww` and height `hh` accordingly with the label size.

- `bool needs_keyboard () const`
  Returns whether this widget needs a keyboard.

- `void needs_keyboard (bool needs)`
  
  Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  Returns if a widget is used for output only.

- `Fi_Group * parent () const`
  
  Returns a pointer to the parent widget.

- `void parent (Fi_Group *p)`
  Internal use only - "for hacks only".

- `void position (int X, int Y)`
  Repositions the window or widget.

- `void redraw ()`
  Schedules the drawing of the widget.

- `void redraw_label ()`
  Schedules the drawing of the label.

- `Fi_Color selection_color () const`
  Gets the selection color.

- `void selection_color (Fi_Color a)`
  Sets the selection color.

- `void set_active ()`
  Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  Marks the value of the widget as changed.

- `void set_output ()`
  Sets a widget to output only.
void set_visible()

Makes the widget visible.

void set_visible_focus()

Enables keyboard focus navigation with this widget.

int shortcut_label() const

Returns whether the widget's label uses '&' to indicate shortcuts.

void shortcut_label(int value)

Sets whether the widget's label uses '&' to indicate shortcuts.

virtual void show()

Makes a widget visible.

void size(int W, int H)

Changes the size of the widget.

int take_focus()

Gives the widget the keyboard focus.

unsigned int takesevents() const

Returns if the widget is able to take events.

int test_shortcut()

Returns true if the widget's label contains the entered '&x' shortcut.

const char* tooltip() const

Gets the current tooltip text.

void tooltip(const char* text)

Sets the current tooltip text.

Fl_Window* top_window()

Returns a pointer to the top-level window for the widget.

Fl_Window* top_window_offset(int &xoff, int &yoff) const

Finds the x/y offset of the current widget relative to the top-level window.

uchar type() const

Gets the widget type.

void type(uchar t)

Sets the widget type.

int use_accents_menu()

Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

void* user_data() const

Gets the user data for this widget.

void user_data(Fl_Callback_User_Data*v, bool auto_free)

Sets the user data for this widget.

void user_data(void*v)

Sets the user data for this widget.

unsigned int visible() const

Returns whether a widget is visible.

unsigned int visible_focus() const

Checks whether this widget has a visible focus.

void visible_focus(int v)

Modifies keyboard focus navigation.

int visible_r() const

Returns whether a widget and all its parents are visible.

int w() const

Gets the widget width.

Fl_When when() const

Returns the conditions under which the callback is called.

void when(uchar i)
Sets the flags used to decide when a callback is called.

- `FL_Window * window () const`
  Returns a pointer to the nearest parent window up the widget hierarchy.
- `int x () const`
  Gets the widget position in its window.
- `int y () const`
  Gets the widget position in its window.
- `virtual ~FL_Widget ()`
  Destroys the widget.

Protected Member Functions

- `virtual int inp_h () const`
  See `inp_x()` for info.
- `virtual int inp_w () const`
  See `inp_x()` for info.
- `virtual int inp_x () const`
  The methods `inp_x()`, `inp_y()`, `inp_w()` and `inp_h()` return the desired position and size of the internal `Fl_Input` widget.
- `virtual int inp_y () const`
  See `inp_x()` for info.
- `virtual int menu_h () const`
  See `menu_x()` for info.
- `virtual int menu_w () const`
  See `menu_x()` for info.
- `virtual int menu_x () const`
  The methods `menu_x()`, `menu_y()`, `menu_w()` and `menu_h()` return the desired position and size of the internal `Fl_Menu_Button` widget.
- `virtual int menu_y () const`
  See `menu_x()` for info.

Protected Member Functions inherited from `Fl_Group`

- `Fl_Rect * bounds ()`
  Returns the internal array of widget sizes and positions.
- `void draw () FL_OVERRIDE`
  Draws the widget.
- `void draw_child (Fl_Widget &widget) const`
  Forces a child to redraw.
- `void draw_children ()`
  Draws all children of the group.
- `void draw_outside_label (const Fl_Widget &widget) const`
  Parents normally call this to draw outside labels of child widgets.
- `virtual int on_insert (Fl_Widget *, int)`
  Allow derived groups to act when a widget is added as a child.
- `virtual int on_move (int, int)`
  Allow derived groups to act when a widget is moved within the group.
- `virtual void on_remove (int)`
  Allow derived groups to act when a child widget is removed from the group.
- `int * sizes ()`
  Returns the internal array of widget sizes and positions.
- `void update_child (Fl_Widget &widget) const`
  Draws a child only if it needs it.
Protected Member Functions inherited from Fl_Widget

- void clear_flag(unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop() const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box() const
  Draws the widget box according its box style.
- void draw_box(Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus() const
  Draws a focus rectangle around the widget.
- void draw_focus(Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
- void draw_focus(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void draw_label() const
  Draws the widget's label at the defined label position.
- void draw_label(int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget(int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.
- unsigned int flags() const
  Gets the widget flags mask.
- void h(int v)
  Internal use only.
- void set_flag(unsigned int c)
  Sets a flag in the flags mask.
- void w(int v)
  Internal use only.
- void x(int v)
  Internal use only.
- void y(int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group ∗current()
  Returns the currently active group.
- static void current(Fl_Group ∗g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback(Fl_Widget ∗widget, void ∗data)
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut(const char ∗t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut(const char ∗, const bool require_alt=false)
  Returns true if the given text ∗t contains the entered '&x' shortcut.
Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPY_LABEL = 1<<10, CLIPCHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  FULLSCREEN = 1<<16, COPY_TOOLTIP = 1<<17, MAC_USE_ACCENTS_MENU = 1<<18,
  NEEDS_KEYBOARD = 1<<19, IMAGE_BOUND = 1<<20, DEIMAGE_BOUND = 1<<21, AUTO_DELETE_USER_DATA = 1<<22,
  MAXIMIZED = 1<<23, USERFLAG3 = 1<<24, USERFLAG2 = 1<<25, USERFLAG1 = 1<<26
}
flags possible values enumeration.

12.76.1 Detailed Description

A combination of the input widget and a menu button.

The user can either type into the input area, or use the menu button chooser on the right to choose an item which loads the input area with the selected text.

The application can directly access both the internal Fl_Input and Fl_Menu_Button widgets respectively using the input() and menubutton() accessor methods.

The default behavior is to invoke the Fl_Input_Choice::callback() if the user changes the input field's contents, either by typing, pasting, or clicking a different item in the choice menu.

The callback can determine if an item was picked vs. typing into the input field by checking the value of menubutton()->changed(), which will be:

- 1: the user picked a different item in the choice menu
- 0: the user typed or pasted directly into the input field

Example Use of Fl_Input_Choice

```c
#include <stdio.h>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Input_Choice.H>

// Fl_Input_Choice callback()
void choice_cb(Fl_Widget *w, void *userdata) {
  // Show info about the picked item
  Fl_Input_Choice *choice = (Fl_Input_Choice*)w;
  printf("*** Choice Callback:\n");
  printf(" widget's text value='%s', choice->value()\n", choice->value()); // normally all you need
  // Access the menu via menubutton().
  const Fl_Menu_Item *item = choice->menubutton()->mvalue();
  printf(" item label()='%s', choice->menubutton()->value()\n", item ? item->label() : "(No item)";
  printf(" input value()='%s', choice->input()->value()\n", item ? item->value() : "(No item)";
  printf(" The user %s\n", choice->menubutton()->changed() ? "picked a menu item" :
            "typed text");
}

int main() {
  Fl_Double_Window win(200,100,"Input Choice");
  win.begin();
  Fl_Input_Choice choice(10,10,100,30);
```
choice.callback(choice_cb, 0);
choice.add("Red");
choice.add("Orange");
choice.add("Yellow");
//choice.value("Red"); // uncomment to make "Red" default
win.end();
win.show();
return Fl::run();
}

Subclassing Example

One can subclass Fl_Input_Choice to override the virtual methods inp_x/y/w/h() and menu_x/y/w/h() to take
control of the internal Fl_Input and Fl_Menu_Button widget positioning. In this example, input and menubutton-
's positions are swapped:

```cpp
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Input_Choice.H>

class MyInputChoice : public Fl_Input_Choice {
protected:
  virtual int inp_x() const { return x() + Fl::box_dx(box()) + menu_w(); } // override to reposition
  virtual int menu_x() const { return x() + Fl::box_dx(box()); } // override to reposition
public:
  MyInputChoice(int X, int Y, int W, int H, const char*L=0) : Fl_Input_Choice(X,Y,W,H,L) {
    resize(X,Y,W,H); // necessary for ctor to trigger our overrides
  }
};

int main(int argc, char **argv) {
  Fl_Window *win = new Fl_Window(400,300);
  MyInputChoice *mychoice = new MyInputChoice(150,40,150,25,"Right Align Input");
  mychoice->add("Aaa");
  mychoice->add("Bbb");
  mychoice->add("Ccc");
  win->end();
  win->resizable(win);
  win->show();
  return Fl::run();
}
```

12.76.2 Constructor & Destructor Documentation

12.76.2.1 Fl_Input_Choice()

Fl_Input_Choice::Fl_Input_Choice {
  int X,
  int Y,
  int W,
  int H,
  const char * L = 0 )
Creates a new Fl_Input_Choice widget using the given position, size, and label string.
Inherited destructor destroys the widget and any values associated with it.

12.76.3 Member Function Documentation

12.76.3.1 add()  

```cpp
void Fl_Input_Choice::add ( 
  const char * s ) [inline]
```

Adds an item to the menu.
When any item is selected, the Fl_Input_Choice callback() is invoked, which can do something with the selected
item.
You can access the more complex Fl_Menu_Button::add() methods (setting item-specific callbacks, userdata, etc),
via menubutton(). Example:

```cpp
Fl_Input_Choice *choice = new Fl_Input_Choice(100,10,120,25,"Fonts");
Fl_Menu_Button *mb = choice->menubutton(); // use Fl_Input_Choice's Fl_Menu_Button
mb->add("Helvetica", 0, MyFont_CB, (void*)mydata); // use Fl_Menu_Button's add() methods
mb->add("Courier", 0, MyFont_CB, (void*)mydata);
mb->add("More..", 0, FontDialog_CB, (void*)mydata);
```
12.76.3.2 inp_x()

virtual int Fl_Input_Choice::inp_x() const [inline], [protected], [virtual]
The methods inp_x(), inp_y(), inp_w() and inp_h() return the desired position and size of the internal Fl_Input widget. These can be overridden by a subclass to redefine positioning. See code example in the Description for subclassing details.

12.76.3.3 input()

Fl_Input * Fl_Input_Choice::input() [inline]
Returns a pointer to the internal Fl_Input widget. This can be used to directly access all of the Fl_Input widget's methods.

12.76.3.4 menu_x()

virtual int Fl_Input_Choice::menu_x() const [inline], [protected], [virtual]
The methods menu_x(), menu_y(), menu_w() and menu_h() return the desired position and size of the internal Fl_Menu_Button widget. These can be overridden by a subclass to redefine positioning. See code example in the Description for subclassing details.

12.76.3.5 menubutton()

Fl_Menu_Button * Fl_Input_Choice::menubutton() [inline]
Returns a pointer to the internal Fl_Menu_Button widget. This can be used to access any of the methods of the menu button, e.g. Fl_Input_Choice * choice = new Fl_Input_Choice(100,10,120,25,"Choice:"); [...] // Print all the items in the choice menu for ( int t=0; t<choice->menubutton()->size(); t++ ) { const Fl_Menu_Item &item = choice->menubutton()->menu()[t]; printf("item %d -- label=%s\n", t, item.label() ? item.label() : "(Null)")); }

12.76.3.6 resize()

void Fl_Input_Choice::resize ( int X, int Y, int W, int H ) [virtual]
Resizes the Fl_Input_Choice widget. Reimplemented from Fl_Group.

12.76.3.7 update_menubutton()

int Fl_Input_Choice::update_menubutton() {}
Updates the menubutton with the string value in Fl_Input. If the string value currently in Fl_Input matches one of the menu items in menubutton(), that menu item will become the current item selected. Call this method after setting value(const char*) if you need the menubutton() to be synchronized with the Fl_Input field. // Add items choice->add(".25"); choice->add(".50"); choice->add("1.0"); choice->add("2.0"); choice->add("4.0"); choice->value("1.0"); // sets Fl_Input to "1.0" choice->update_menubutton(); // cause menubutton to reflect this value too // [returns 1 if match was found, 0 if not] // Verify menubutton()'s value.
printf("menu button choice index=%d, value=%s\n",
   choice->menubutton()->value(), // would be -1 if update not done
   choice->menubutton()->text()); // would be NULL if update not done

Generated by Doxygen
Returns

1 if a matching menuitem was found and value set, 0 if not.

Version

1.4.0

12.76.3.8 value() [1/2]

void Fl_Input_Choice::value (const char ∗val) [inline]

Sets the Fl_Input text field's contents to val.

Note it is possible to set the value() to one that is not in the menubutton's list of choices.

Setting the value() does NOT affect the menubutton's selection. If that's needed, call update_menubutton() after setting value().

See also

void value(int val), update_menubutton()

12.76.3.9 value() [2/2]

void Fl_Input_Choice::value (int val)

Chooses item# val in the menu, and sets the Fl_Input text field to that value.

Any previous text is cleared.

See also

void value(const char ∗val)

The documentation for this class was generated from the following files:

- Fl_Input_Choice.H
- Fl_Input_Choice.cxx

12.77 Fl_Int_Input Class Reference

The Fl_Int_Input class is a subclass of Fl_Input that only allows the user to type decimal digits (or hex numbers of the form 0xabcdef).

#include <Fl_Int_Input.H>

Inheritance diagram for Fl_Int_Input:

```
Fl_Widget
  ↓
Fl_Input_
  ↓
Fl_Input
  ↓
Fl_Int_Input
```

Public Member Functions

- Fl_Int_Input (int X, int Y, int W, int H, const char ∗l=0)

  Creates a new Fl_Int_Input widget using the given position, size, and label string.
Public Member Functions inherited from Fl_Input

- Fl_Input (int, int, int, int, const char ∗=0)
  Creates a new Fl_Input widget using the given position, size, and label string.
- int handle (int) FL_OVERRIDE
  Handles the specified event.

Public Member Functions inherited from Fl_Input_

- int append (const char ∗t, int l=0, char keep_selection=0)
  Append text at the end.
- bool can_redo () const
  Check if there is a redo action available.
- bool can_undo () const
  Check if the last operation can be undone.
- int copy (int clipboard)
  Put the current selection into the clipboard.
- int copy_cuts ()
  Copies the yank buffer to the clipboard.
- Fl_Color cursor_color () const
  Gets the color of the cursor.
- void cursor_color (Fl_Color n)
  Sets the color of the cursor.
- int cut ()
  Deletes the current selection.
- int cut (int a, int b)
  Deletes all characters between index \( a \) and \( b \).
- int cut (int n)
  Deletes the next \( n \) bytes rounded to characters before or after the cursor.
- double dvalue () const
  Returns the widget text interpreted as a floating point number.
- Fl_Input_ (int, int, int, int, const char ∗=0)
  Creates a new Fl_Input_ widget.
- unsigned int index (int i) const
  Returns the character at index \( i \).
- int input_type () const
  Gets the input field type.
- void input_type (int t)
  Sets the input field type.
- int insert (const char ∗t, int l=0)
  Inserts text at the cursor position.
- int insert_position () const
  Gets the position of the text cursor.
- int insert_position (int p)
  Sets the cursor position and mark.
- int insert_position (int p, int m)
  Sets the index for the cursor and mark.
- int ivalue () const
  Returns the widget text interpreted as a signed integer.
- int mark () const
  Gets the current selection mark.
- int mark (int m)
Sets the current selection mark.

- `int maximum_size () const`
  
  Gets the maximum length of the input field in characters.

- `void maximum_size (int m)`
  
  Sets the maximum length of the input field in characters.

- `int position () const`
- `int position (int p)`
- `int position (int p, int m)`
- `int readonly () const`
  
  Gets the read-only state of the input field.

- `void readonly (int b)`
  
  Sets the read-only state of the input field.

- `int redo ()`
  
  Redo previous undo operation.

- `int replace (int b, int e, const char ∗text, int ilen=0)`
  
  Deletes text from b to e and inserts the new string text.

- `void resize (int, int, int, int) FL_OVERRIDE`
  
  Changes the size of the widget.

- `int shortcut () const`
  
  Returns the shortcut key associated with this widget.

- `void shortcut (int s)`
  
  Sets the shortcut key associated with this widget.

- `int size () const`
  
  Returns the number of bytes in value().

- `void size (int W, int H)`
  
  Sets the width and height of this widget.

- `int static_value (const char ∗)`
  
  Changes the widget text.

- `int static_value (const char ∗, int)`
  
  Changes the widget text.

- `int tab_nav () const`
  
  Gets whether the Tab key causes focus navigation in multiline input fields or not.

- `void tab_nav (int val)`
  
  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.

- `Fl_Color textcolor () const`
  
  Gets the color of the text in the input field.

- `void textcolor (Fl_Color n)`
  
  Sets the color of the text in the input field.

- `Fl_Font textfont () const`
  
  Gets the font of the text in the input field.

- `void textfont (Fl_Font s)`
  
  Sets the font of the text in the input field.

- `Fl_Fontsize textsize () const`
  
  Gets the size of the text in the input field.

- `void textsize (Fl_Fontsize s)`
  
  Sets the size of the text in the input field.

- `int undo ()`
  
  Undoes previous changes to the text buffer.

- `const char ∗value () const`
  
  Returns the text displayed in the widget.
Changes the widget text.
• int value (const char *, int)

Changes the widget text.
• int value (double value)

Changes the widget text to a floating point number ("%g").
• int value (int value)

Changes the widget text to a signed integer number.
• int wrap () const

Gets the word wrapping state of the input field.
• void wrap (int b)

Sets the word wrapping state of the input field.
• ~Fl_Input_ ()

Destroys the widget.

Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()

• void _set_fullscreen ()

• void activate ()

Activates the widget.
• unsigned int active () const

Returns whether the widget is active.
• int active_r () const

Returns whether the widget and all of its parents are active.
• Fl_Align align () const

Gets the label alignment.
• void align (Fl_Align alignment)

Sets the label alignment.
• long argument () const

Gets the current user data (long) argument that is passed to the callback function.
• void argument (long v)

Sets the current user data (long) argument that is passed to the callback function.
• virtual class Fl_Gl_Window * as_gl_window ()

Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const * as_gl_window () const

• virtual Fl_Group * as_group ()

Returns an Fl_Group pointer if this widget is an Fl_Group.
• virtual Fl_Group const * as_group () const

• virtual Fl_Window * as_window ()

Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const * as_window () const

• void bind_deimage (Fl_Image * img)

Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage (int f)

Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image (Fl_Image * img)

Sets the image to use as part of the widget label when in the active state.
• void bind_image (int f)

Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• Fl_Boxtype box () const

Gets the box type of the widget.
• void box (Fl_Boxtype new_box)
  
  Sets the box type for the widget.

• Fl_Callback_p callback () const
  
  Gets the current callback function for the widget.

• void callback (Fl_Callback ∗cb)
  
  Sets the current callback function for the widget.

• void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)
  
  Sets the current callback function and managed user data for the widget.

• void callback (Fl_Callback ∗cb, void ∗p)
  
  Sets the current callback function and data for the widget.

• void callback (Fl_Callback0 ∗cb)
  
  Sets the current callback function for the widget.

• void callback (Fl_Callback1 ∗cb, long p=0)
  
  Sets the current callback function for the widget.

• unsigned int changed () const
  
  Checks if the widget value changed since the last callback.

• void clear_active ()
  
  Marks the widget as inactive without sending events or changing focus.

• void clear_changed ()
  
  Marks the value of the widget as unchanged.

• void clear_damage (uchar c=0)
  
  Clears or sets the damage flags.

• void clear_output ()
  
  Sets a widget to accept input.

• void clear_visible ()
  
  Hides the widget.

• void clear_visible_focuss ()
  
  Disables keyboard focus navigation with this widget.

• Fl_Color color () const
  
  Gets the background color of the widget.

• void color (Fl_Color bg)
  
  Sets the background color of the widget.

• void color (Fl_Color bg, Fl_Color sel)
  
  Sets the background and selection color of the widget.

• Fl_Color color2 () const
  
  For back compatibility only.

• void color2 (unsigned a)
  
  For back compatibility only.

• int contains (const Fl_Widget ∗w) const
  
  Checks if w is a child of this widget.

• void copy_label (const char ∗new_label)
  
  Sets the current label.

• void copy_tooltip (const char ∗text)
  
  Sets the current tooltip text.

• uchar damage () const
  
  Returns non-zero if draw() needs to be called.

• void damage (uchar c)
  
  Sets the damage bits for the widget.

• void damage (uchar c, int x, int y, int w, int h)
  
  Sets the damage bits for an area inside the widget.

• int damage_resize (int, int, int, int)
Internal use only.

- **void deactivate ()**
  Deactivates the widget.

- **Fl_Image ∗ deimage ()**
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image ∗ deimage () const**
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (Fl_Image ∗img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (Fl_Image &img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const**
  Returns whether the inactive image is managed by the widget.

- **void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with default arguments.

- **void do_callback (Fl_Widget ∗wgt, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void do_callback (Fl_Widget ∗wgt, void ∗arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void draw_label (int, int, int, int, Fl_Align) const**
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h () const**
  Gets the widget height.

- **virtual void hide ()**
  Makes a widget invisible.

- **Fl_Image ∗ image ()**
  Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image ∗ image () const**
  Gets the image that is used as part of the widget label when in the active state.

- **void image (Fl_Image &img)**
  Sets the image to use as part of the widget label when in the active state.

- **void image (Fl_Image ∗img)**
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const**
  Returns whether the image is managed by the widget.

- **int inside (const Fl_Widget ∗wgt) const**
  Checks if this widget is a child of wgt.

- **int is_label_copied () const**
  Returns whether the current label was assigned with copy_label().

- **const char ∗ label () const**
  Gets the current label text.

- **void label (const char ∗text)**
  Sets the current label pointer.

- **void label (Fl_Labeltype a, const char ∗b)**
  Shortcut to set the label text and type in one call.

- **Fl_Color labelcolor () const**
  Gets the label color.

- **void labelcolor (Fl_Color c)**
  Sets the label color.

- **Fl_Font labelfont () const**
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group * p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
Gives the widget the keyboard focus.

- **unsigned int takesevents () const**
  
  Returns if the widget is able to take events.

- **int test_shortcut ()**
  
  Returns true if the widget’s label contains the entered ‘&x’ shortcut.

- **const char * tooltip () const**
  
  Gets the current tooltip text.

- **void tooltip (const char *text)**
  
  Sets the current tooltip text.

- **Fl_Window * top_window () const**
  
  Returns a pointer to the top-level window for the widget.

- **Fl_Window * top_window_offset (int &xoff, int &yoff) const**
  
  Finds the x/y offset of the current widget relative to the top-level window.

- **uchar type () const**
  
  Gets the widget type.

- **void type (uchar t)**
  
  Sets the widget type.

- **int use_accents_menu ()**
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- **void * user_data () const**
  
  Gets the user data for this widget.

- **void user_data (Fl_Callback_User_Data *v, bool auto_free)**
  
  Sets the user data for this widget.

- **void user_data (void *v)**
  
  Sets the user data for this widget.

- **unsigned int visible () const**
  
  Returns whether a widget is visible.

- **unsigned int visible_focus () const**
  
  Checks whether this widget has a visible focus.

- **void visible_focus (int v)**
  
  Modifies keyboard focus navigation.

- **int visible_r () const**
  
  Returns whether a widget and all its parents are visible.

- **int w () const**
  
  Gets the widget width.

- **Fl_When when () const**
  
  Returns the conditions under which the callback is called.

- **void when (uchar i)**
  
  Sets the flags used to decide when a callback is called.

- **Fl_Window * window () const**
  
  Returns a pointer to the nearest parent window up the widget hierarchy.

- **int x () const**
  
  Gets the widget position in its window.

- **int y () const**
  
  Gets the widget position in its window.

- **virtual ~Fl_Widget ()**
  
  Destroys the widget.
Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don’t set a callback.
- static unsigned int labelShortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int testShortcut (const char *t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Static Public Attributes inherited from Fl_Input

- static const char *copy_menu_text = "Copy"
  [this text may be customized at run-time]
- static const char *cut_menu_text = "Cut"
  [this text may be customized at run-time]
- static const char *paste_menu_text = "Paste"
  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
  }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Input

- void draw () FL_OVERRIDE
  Draws the widget.
- int handle_key ()
  Handles a keystroke.
- int handle_rmb ()
  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

- int apply_undo ()
  Apply the current undo/redo operation.
- void drawtext (int, int, int, int)
  Draws the text in the passed bounding box.
- void drawtext (int, int, int, int, bool draw_active)
  Draws the text in the passed bounding box.
- void handle_mouse (int, int, int, int keepmark=0)
Handles mouse clicks and mouse moves.

- int **handle_text**(int e, int, int, int, int)

Handles all kinds of text field related events.

- int **line_end**(int i) const

  Finds the end of a line.

- int **line_start**(int i) const

  Finds the start of a line.

- int **linesPerPage**

- void **maybe_do_callback**(Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

- int **up_down_position**(int, int keepmark=0)

  Moves the cursor to the column given by up_down_pos.

- int **word_end**(int i) const

  Finds the end of a word.

- int **word_start**(int i) const

  Finds the start of a word.

- int **xscroll**

- int **y_scroll**

- void **yscroll**(int yOffset)

Protected Member Functions inherited from Fl_Widget

- void **clear_flag**(unsigned int c)

  Clears a flag in the flags mask.

- void **draw_backdrop**(const)

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void **draw_box**(const)

  Draws the widget box according its box style.

- void **draw_box**(Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.

- void **draw_box**(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.

- void **draw_focus**(const)

  Draws a focus rectangle around the widget.

- void **draw_focus**(Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.

- void **draw_focus**(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.

- void **draw_label**(const)

  Draws the widget's label at the defined label position.

- void **draw_label**(int, int, int, int) const

  Draws the label in an arbitrary bounding box.

- Fl_Widget**(int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.

- unsigned int **flags**(const)

  Gets the widget flags mask.

- void **h**(int v)

  Internal use only.

- void **set_flag**(unsigned int c)

  Sets a flag in the flags mask.

- void **w**(int v)

  Internal use only.
• void x (int v)
  Internal use only.
• void y (int v)
  Internal use only.

12.77.1 Detailed Description

The Fl_Int_Input class is a subclass of Fl_Input that only allows the user to type decimal digits (or hex numbers of the form 0xae).

12.77.2 Constructor & Destructor Documentation

12.77.2.1 Fl_Int_Input()

Fl_Int_Input::Fl_Int_Input (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char ∗ l = 0 )

Creates a new Fl_Int_Input widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.
Inherited destructor destroys the widget and any value associated with it.
The documentation for this class was generated from the following files:

• Fl_Int_Input.H
• Fl_Input.cxx

12.78 Fl_JPEG_Image Class Reference

The Fl_JPEG_Image class supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG) File Interchange Format (JFIF) images.

#include <Fl_JPEG_Image.H>

Inheritance diagram for Fl_JPEG_Image:

Fl_JPEG_Image
Fl_RGB_Image
Fl_Image

Public Member Functions

• Fl_JPEG_Image (const char ∗filename)
  The constructor loads the JPEG image from the given jpeg filename.
• Fl_JPEG_Image (const char ∗name, const unsigned char ∗data, int data_length=-1)
  The constructor loads the JPEG image from memory.

Public Member Functions inherited from Fl_RGB_Image

• virtual Fl_SVG_Image ∗ as_svg_image ()

Generated by Doxygen
• void color_average (Fl_Color c, float i) FL_OVERRIDE
  The color_average() method averages the colors in the image with the provided FLTK color value.
• Fl_Image * copy () const
  Creates a resized copy of the image.
• void desaturate () FL_OVERRIDE
  The desaturate() method converts an image to grayscale.
• void draw (int X, int Y)
• void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE
  Draws the image to the current drawing surface with a bounding box.
• Fl_RGB_Image (const Fl_Pixmap * pxm, Fl_Color bg=FL_GRAY)
  The constructor creates a new RGBA image from the specified Fl_Pixmap.
• Fl_RGB_Image (const uchar * bits, int bits_length, int W, int H, int D, int LD)
  The constructor creates a new image from the specified data.
• Fl_RGB_Image (const uchar * bits, int W, int H, int D=3, int LD=0)
  The constructor creates a new image from the specified data.
• void label (Fl_Menu_Item * m) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a menu item.
• void label (Fl_Widget * w) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a widget or menu item.
• virtual void normalize ()
  Makes sure the object is fully initialized.
• void uncache () FL_OVERRIDE
  If the image has been cached for display, delete the cache data.
• ~Fl_RGB_Image () FL_OVERRIDE
  The destructor frees all memory and server resources that are used by the image.

Public Member Functions inherited from Fl_Image

• virtual class Fl_Shared_Image * as_shared_image ()
  Returns whether an image is an Fl_Shared_Image or not.
• Fl_Image * copy () const
  Creates a copy of the image in the same size.
• int count () const
  Returns the number of data values associated with the image.
• int d () const
  Returns the image depth.
• const char *const * data () const
  Returns a pointer to the current image data array.
• int data_h () const
  Returns the height of the image data.
• int data_w () const
  Returns the width of the image data.
• void draw (int X, int Y)
  Draws the image to the current drawing surface.
• int fail () const
  Returns a value that is not 0 if there is currently no image available.
• Fl_Image (int W, int H, int D)
  The constructor creates an empty image with the specified width, height, and depth.
• int h () const
  Returns the current image drawing height in FLTK units.
• void **inactive** ()
  The **inactive()** method calls **color_average(FL_BACKGROUND_COLOR, 0.33f)** to produce an image that appears
grayed out.

• int **ld** () const
  Returns the current line data size in bytes.

• virtual void **release** ()
  Releases an **Fl_Image** - the same as 'delete this'.

• virtual void **scale** (int width, int height, int proportional=1, int can_expand=0)
  Sets the drawing size of the image.

• int **w** () const
  Returns the current image drawing width in FLTK units.

• virtual **~Fl_Image** ()
  The destructor is a virtual method that frees all memory used by the image.

Protected Member Functions

• void **load_jpg_** (const char *filename, const char *sharename, const unsigned char *data, int data_length=-1)

Protected Member Functions inherited from Fl_Image

• void **d** (int D)
  Sets the current image depth.

• void **data** (const char *const *p, int c)
  Sets the current data pointer and count of pointers in the array.

• void **draw_empty** (int X, int Y)
  The protected method **draw_empty()** draws a box with an X in it.

• int **draw_scaled** (int X, int Y, int W, int H)
  Draw the image to the current drawing surface rescaled to a given width and height.

• void **h** (int H)
  Sets the height of the image data.

• void **Id** (int LD)
  Sets the current line data size in bytes.

• void **w** (int W)
  Sets the width of the image data.

Additional Inherited Members

Static Public Member Functions inherited from Fl_RGB_Image

• static size_t **max_size** ()
  Returns the maximum allowed image size in bytes when creating an **Fl_RGB_Image** object.

• static void **max_size** (size_t size)
  Sets the maximum allowed image size in bytes when creating an **Fl_RGB_Image** object.

Static Public Member Functions inherited from Fl_Image

• static **Fl_Labeltype** **define_FL_IMAGE_LABEL** ()
• static **Fl_RGB_Scaling** **RGB_scaling** ()
  Returns the currently used RGB image scaling method.

• static void **RGB_scaling** (Fl_RGB_Scaling)
  Sets the RGB image scaling method used for copy(int, int).

• static **Fl_RGB_Scaling** **scaling_algorithm** ()
  Gets what algorithm is used when resizing a source image to draw it.

• static void **scaling_algorithm** (Fl_RGB_Scaling algorithm)
  Sets what algorithm is used when resizing a source image to draw it.
Public Attributes inherited from Fl_RGB_Image

- int alloc_array
  If non-zero, the object's data array is delete[]d when deleting the object.
- const uchar * array
  Points to the start of the object's data array.

Static Public Attributes inherited from Fl_Image

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_MEMORY_ACCESS = -4
- static const int ERR_NO_IMAGE = -1
- static bool register_images_done = false
  True after fl_register_images() was called, false before.

Static Protected Member Functions inherited from Fl_Image

- static void labeltype (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void measure (const Fl_Label *lo, int &lw, int &lh)

12.78.1 Detailed Description

The Fl_JPEG_Image class supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG) File Interchange Format (JFIF) images. The class supports grayscale and color (RGB) JPEG image files.

12.78.2 Constructor & Destructor Documentation

12.78.2.1 Fl_JPEG_Image()

Fl_JPEG_Image::Fl_JPEG_Image (const char * filename )
const char * filename )
The constructor loads the JPEG image from the given jpeg filename.
The inherited destructor frees all memory and server resources that are used by the image.
Use Fl_Image::fail() to check if Fl_JPEG_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the JPEG format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason. If the image has loaded correctly, w(), h(), and d() should return values greater than zero.

Parameters

- filename [in] a full path and name pointing to a valid jpeg file.

See also

Fl_JPEG_Image::Fl_JPEG_Image(const char * imagename, const unsigned char * data)

12.78.2.2 Fl_JPEG_Image()

Fl_JPEG_Image::Fl_JPEG_Image (const char * name,
const unsigned char * data,
int data_length = -1 )
The constructor loads the JPEG image from memory.
Construct an image from a block of memory inside the application. Fluid offers "binary Data" chunks as a great way to add image data into the C++ source code. name_png can be NULL. If a name is given, the image is added to the list of shared images (see: Fl_Shared_Image) and will be available by that name.
The inherited destructor frees all memory and server resources that are used by the image. Use Fl_Image::fail() to check if Fl_JPEG_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the JPEG format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason. If the image has loaded correctly, w(), h(), and d() should return values greater than zero.

Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>A unique name or NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>A pointer to the memory location of the JPEG image</td>
</tr>
<tr>
<td>data_length</td>
<td>optional length of data. This will protect memory outside of the data array from illegal read operations</td>
</tr>
</tbody>
</table>

See also

Fl_JPEG_Image::Fl_JPEG_Image(const char *filename)
Fl_Shared_Image

The documentation for this class was generated from the following files:

- Fl_JPEG_Image.H
- Fl_JPEG_Image.cxx

12.79 Fl_Label Struct Reference

This struct stores all information for a text or mixed graphics label.

#include <Fl_Widget.H>

Public Member Functions

- void draw (int, int, int, int, Fl_Align) const

  Draws the label aligned to the given box.

- void measure (int &w, int &h) const

  Measures the size of the label.

Public Attributes

- Fl_Align align_

  alignment of label

- Fl_Color color

  text color

- Fl_Image *deimage

  optional image for a deactivated label

- Fl_Font font

  label font used in text

- Fl_Image *image

  optional image for an active label

- Fl_Fontsize size

  size of label font

- uchar type

  type of label.

- const char *value

  label text
12.79.1 Detailed Description
This struct stores all information for a text or mixed graphics label.

Todo There is an aspiration that the Fl_Label type will become a widget by itself. That way we will be avoiding a
lot of code duplication by handling labels in a similar fashion to widgets containing text. We also provide an
easy interface for very complex labels, containing html or vector graphics. However, this re-factoring is not in
place in this release.

12.79.2 Member Function Documentation
12.79.2.1 draw()

void Fl_Label::draw (  
    int X,  
    int Y,  
    int W,  
    int H,  
    Fl_Align align ) const

Draws the label aligned to the given box.
Draws a label with arbitrary alignment in an arbitrary box.

12.79.2.2 measure()

void Fl_Label::measure (  
    int & W,  
    int & H ) const

Measures the size of the label.

Parameters

| in, out | W,H | : this is the requested size for the label text plus image; on return, this will contain the size |
|---------|-----|: this is the requested size for the label text plus image; on return, this will contain the size |
|         |     | needed to fit the label |

12.79.3 Member Data Documentation
12.79.3.1 type

uchar Fl_Label::type

type of label.

See also

Fl_Labeltype

The documentation for this struct was generated from the following files:

- Fl_Widget.H
- fl_labeltype.cxx

12.80 Fl_Light_Button Class Reference

This subclass displays the "on" state by turning on a light, rather than drawing pushed in.

#include <Fl_Light_Button.H>

Inheritance diagram for Fl_Light_Button:
Public Member Functions

- **Fl_Light_Button (int x, int y, int w, int h, const char ∗=0)**
  
  Creates a new Fl_Light_Button widget using the given position, size, and label string.

- **int handle (int) FL_OVERRIDE**
  
  Handles the specified event.

Public Member Functions inherited from Fl_Button

- **int clear ()**
  
  Same as value(0).

- **uchar compact ()**
  
  Return true if buttons are rendered as compact buttons.

- **void compact (uchar v)**
  
  Decide if buttons should be rendered in compact mode.

- **Fl_Boxtype down_box () const**
  
  Returns the current down box type, which is drawn when value() is non-zero.

- **void down_box (Fl_Boxtype b)**
  
  Sets the down box type.

- **Fl_Color down_color () const**
  
  (for backwards compatibility)

- **void down_color (unsigned c)**
  
  (for backwards compatibility)

- **Fl_Button (int X, int Y, int W, int H, const char ∗L=0)**
  
  The constructor creates the button using the given position, size, and label.

- **int set ()**
  
  Same as value(1).

- **void setonly ()**
  
  Turns on this button and turns off all other radio buttons in the group (calling value(1) or set() does not do this).

- **int shortcut () const**
  
  Returns the current shortcut key for the button.

- **void shortcut (const char ∗s)**
  
  (for backwards compatibility)

- **void shortcut (int s)**
  
  Sets the shortcut key to s.

- **char value () const**
  
  Returns the current value of the button (0 or 1).

- **int value (int v)**
  
  Sets the current value of the button.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ as_gl_window () const
- virtual Fl_Group ∗ as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ as_group () const
- virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ as_window () const
- void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
Checks if the widget value changed since the last callback.

- **void clear_active ()**
  
  Marks the widget as inactive without sending events or changing focus.

- **void clear_changed ()**
  
  Marks the value of the widget as unchanged.

- **void clear_damage (uchar c=0)**
  
  Clears or sets the damage flags.

- **void clear_output ()**
  
  Sets a widget to accept input.

- **void clear_visible ()**
  
  Hides the widget.

- **void clear_visible_focus ()**
  
  Disables keyboard focus navigation with this widget.

- **Fl_Color color () const**
  
  Gets the background color of the widget.

- **void color (Fl_Color bg)**
  
  Sets the background color of the widget.

- **void color (Fl_Color bg, Fl_Color sel)**
  
  Sets the background and selection color of the widget.

- **Fl_Color color2 () const**
  
  For back compatibility only.

- **void color2 (unsigned a)**
  
  For back compatibility only.

- **int contains (const Fl_Widget *w) const**
  
  Checks if w is a child of this widget.

- **void copy_label (const char *new_label)**
  
  Sets the current label.

- **void copy_tooltip (const char *text)**
  
  Sets the current tooltip text.

- **uchar damage () const**
  
  Returns non-zero if draw() needs to be called.

- **void damage (uchar c)**
  
  Sets the damage bits for the widget.

- **void damage (uchar c, int x, int y, int w, int h)**
  
  Sets the damage bits for an area inside the widget.

- **int damage_resize (int, int, int, int)**
  
  Internal use only.

- **void deactivate ()**
  
  Deactivates the widget.

- **Fl_Image * deimage ()**
  
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image * deimage () const**
  
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (Fl_Image &img)**
  
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (Fl_Image *img)**
  
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const**
  
  Returns whether the inactive image is managed by the widget.

- **void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int int, int int, int int, Fl_Choice) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image *image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image *image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Label a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
Sets whether this widget needs a keyboard.

- unsigned int output() const
  Returns if a widget is used for output only.

- Fl_Group * parent() const
  Returns a pointer to the parent widget.

- void parent(Fl_Group *p)
  Internal use only - “for hacks only”.

- void position(int X, int Y)
  Repositions the window or widget.

- void redraw()
  Schedules the drawing of the widget.

- void redraw_label()
  Schedules the drawing of the label.

- virtual void resize(int x, int y, int w, int h)
  Changes the size or position of the widget.

- Fl_Color selection_color() const
  Gets the selection color.

- void selection_color(Fl_Color a)
  Sets the selection color.

- void set_active()
  Marks the widget as active without sending events or changing focus.

- void set_changed()
  Marks the value of the widget as changed.

- void set_output()
  Sets a widget to output only.

- void set_visible()
  Makes the widget visible.

- void set_visible_focus()
  Enables keyboard focus navigation with this widget.

- int shortcut_label() const
  Returns whether the widget's label uses '&' to indicate shortcuts.

- void shortcut_label(int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

- virtual void show()
  Makes a widget visible.

- void size(int W, int H)
  Changes the size of the widget.

- int take_focus()
  Gives the widget the keyboard focus.

- unsigned int takesevents() const
  Returns if the widget is able to take events.

- int testShortcut()
  Returns true if the widget's label contains the entered '&x' shortcut.

- const char * tooltip() const
  Gets the current tooltip text.

- void tooltip(const char *text)
  Sets the current tooltip text.

- Fl_Window * top_window()
  Returns a pointer to the top-level window for the widget.

- Fl_Window * top_window_offset(int &xoff, int &yoff)
  Finds the x/y offset of the current widget relative to the top-level window.
• **uchar type () const**
  
  Gets the widget type.

• **void type (uchar t)**
  
  Sets the widget type.

• **int use_accents_menu ()**
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• **void * user_data () const**
  
  Gets the user data for this widget.

• **void user_data (Fl_Callback_User_Data *v, bool auto_free)**
  
  Sets the user data for this widget.

• **void user_data (void *v)**
  
  Sets the user data for this widget.

• **unsigned int visible () const**
  
  Returns whether a widget is visible.

• **unsigned int visible_focus () const**
  
  Checks whether this widget has a visible focus.

• **void visible_focus (int v)**
  
  Modifies keyboard focus navigation.

• **int visible_r () const**
  
  Returns whether a widget and all its parents are visible.

• **int w () const**
  
  Gets the widget width.

• **FL_When when () const**
  
  Returns the conditions under which the callback is called.

• **void when (uchar i)**
  
  Sets the flags used to decide when a callback is called.

• **FL_Window * window () const**
  
  Returns a pointer to the nearest parent window up the widget hierarchy.

• **int x () const**
  
  Gets the widget position in its window.

• **int y () const**
  
  Gets the widget position in its window.

• **virtual ~Fl_Widget ()**
  
  Destroys the widget.

**Protected Member Functions**

• **void draw () FL_OVERRIDE**
  
  Draws the widget.

**Protected Member Functions inherited from Fl_Button**

• **void simulate_key_action ()**

**Protected Member Functions inherited from Fl_Widget**

• **void clear_flag (unsigned int c)**
  
  Clears a flag in the flags mask.

• **void draw_backdrop () const**
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• **void draw_box () const**
  
  Draws the widget box according its box style.
• **void draw_box (Fl_Boxtype t, Fl_Color c) const**
  Draws a box of type `t`, of color `c` at the widget's position and size.

• **void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const**
  Draws a box of type `t`, of color `c` at the position `X,Y` and size `W,H`.

• **void draw_focus () const**
  Draws a focus rectangle around the widget.

• **void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const**
  Draws a focus box for the widget at the given position and size.

• **void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const**
  Draws a focus box for the widget at the given position and size.

• **void draw_label () const**
  Draws the widget's label at the defined label position.

• **void draw_label (int, int, int, int) const**
  Draws the label in an arbitrary bounding box.

• **Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)**
  Creates a widget at the given position and size.

• **unsigned int flags () const**
  Gets the widget flags mask.

• **void h (int v)**
  Internal use only.

• **void set_flag (unsigned int c)**
  Sets a flag in the flags mask.

• **void w (int v)**
  Internal use only.

• **void x (int v)**
  Internal use only.

• **void y (int v)**
  Internal use only.

---

**Additional Inherited Members**

**Static Public Member Functions inherited from Fl_Widget**

• **static void default_callback (Fl_Widget ∗widget, void ∗data)**
  The default callback for all widgets that don't set a callback.

• **static unsigned int label_shortcut (const char ∗t)**
  Returns the Unicode value of the '&x' shortcut in a given text.

• **static int test_shortcut (const char ∗t, const bool require_alt=false)**
  Returns true if the given text `t` contains the entered '&x' shortcut.

---

**Protected Types inherited from Fl_Widget**

• **enum {**
  \[ details of enum \]
  \}
  flags possible values enumeration.

Generated by Doxygen
Static Protected Member Functions inherited from Fl_Button

- static void key_release_timeout (void *)

Static Protected Attributes inherited from Fl_Button

- static Fl_Widget_Tracker * key_release_tracker = 0

12.80.1 Detailed Description

This subclass displays the "on" state by turning on a light, rather than drawing pushed in. The shape of the "light" is initially set to FL_DOWN_BOX. The color of the light when on is controlled with selection_color(), which defaults to FL_YELLOW.

Buttons generate callbacks when they are clicked by the user. You control exactly when and how by changing the values for type() and when().

![Image of Fl_Light_Button](image.png)

Figure 12.27 Fl_Light_Button

12.80.2 Constructor & Destructor Documentation

12.80.2.1 Fl_Light_Button()

Fl_Light_Button::Fl_Light_Button ( int X, int Y, int W, int H, const char * l = 0 )

Creates a new Fl_Light_Button widget using the given position, size, and label string. The default box type is FL_UP_BOX and the default down box type down_box() is FL_NO_BOX (0). The selection_color() sets the color of the "light". Default is FL_YELLOW.

The default label alignment is 'FL_ALIGN_LEFT|FL_ALIGN_INSIDE' so the label is drawn inside the button area right of the "light".

Note

Do not change the default box types of Fl_Light_Button. The box types determine how the button is drawn. If you change the down_box() type the drawing behavior is undefined.

12.80.3 Member Function Documentation

12.80.3.1 draw()

void Fl_Light_Button::draw ( ) [protected], [virtual]

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();
```

Reimplemented from Fl_Button.

12.80.3.2 handle()

int Fl_Light_Button::handle ( int event ) [virtual]
Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in   | event | the kind of event received |

Return values

| 0    | if the event was not used or understood |
| 1    | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Button.
The documentation for this class was generated from the following files:

- Fl_Light_Button.H
- Fl_Light_Button.cxx

12.81 Fl_Line_Dial Class Reference

Inheritance diagram for Fl_Line_Dial:

```
Fl_Widget
   
Fl_Valuator
   
Fl_Dial
   
Fl_Line_Dial
```

Public Member Functions

- Fl_Line_Dial (int X, int Y, int W, int H, const char *L=0)

Public Member Functions inherited from Fl_Dial

- short angle1 () const
  
  Sets Or gets the angles used for the minimum and maximum values.

- void angle1 (short a)
See short `angle1() const`.

- short `angle2 () const`
  See short `angle1() const`.

- void `angle2 (short a)`
  See short `angle1() const`.

- void `angles (short a, short b)`
  See short `angle1() const`.

- `Fl_Dial (int x, int y, int w, int h, const char ∗l=0)`
  Creates a new `Fl_Dial` widget using the given position, size, and label string.

- int `handle (int)` FL_OVERRIDE
  Allow subclasses to handle event based on current position and size.

**Public Member Functions inherited from `Fl_Valuator`**

- void `bounds (double a, double b)`
  Sets the minimum (a) and maximum (b) values for the valuator widget.

- double `clamp (double)`
  Clamps the passed value to the valuator range.

- virtual int `format (char ∗)`
  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.

- double `increment (double, int)`
  Adds n times the step value to the passed value.

- double `maximum () const`
  Gets the maximum value for the valuator.

- void `maximum (double a)`
  Sets the maximum value for the valuator.

- double `minimum () const`
  Gets the minimum value for the valuator.

- void `minimum (double a)`
  Sets the minimum value for the valuator.

- void `precision (int digits)`
  Sets the step value to $1.0 / 10^{digits}$.

- void `range (double a, double b)`
  Sets the minimum and maximum values for the valuator.

- double `round (double)`
  Round the passed value to the nearest step increment.

- double `step () const`
  Gets or sets the step value.

- void `step (double a, int b)`
  See double `Fl_Valuator::step() const`

- void `step (double s)`
  See double `Fl_Valuator::step() const`.

- void `step (int a)`
  See double `Fl_Valuator::step() const`.

- double `value () const`
  Gets the floating point(double) value.

- int `value (double)`
  Sets the current value.
Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  Activates the widget.
• unsigned int active () const
  Returns whether the widget is active.
• int active_r () const
  Returns whether the widget and all of its parents are active.
• Fl_Align align () const
  Gets the label alignment.
• void align (Fl_Align alignment)
  Sets the label alignment.
• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
• virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const ∗ as_gl_window () const
• virtual Fl_Group ∗ as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• virtual Fl_Group const ∗ as_group () const
• virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const ∗ as_window () const
• void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• Fl_Boxtype box () const
  Gets the box type of the widget.
• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
• FL_Callback_p callback () const
  Gets the current callback function for the widget.
• void callback (FL_Callback ∗ cb)
  Sets the current callback function for the widget.
• void callback (FL_Callback ∗ cb, FL_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (FL_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
• void callback (FL_Callback0 ∗ cb)
  Sets the current callback function for the widget.
• void callback (FL_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
Checks if the widget value changed since the last callback.

- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  Clears or sets the damage flags.

- void clear_output ()
  Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.

- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

- Fl_Color color () const
  Gets the background color of the widget.

- void color (Fl_Color bg)
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

- Fl_Color color2 () const
  For back compatibility only.

- void color2 (unsigned a)
  For back compatibility only.

- int contains (const Fl_Widget * w) const
  Checks if w is a child of this widget.

- void copy_label (const char * new_label)
  Sets the current label.

- void copy_tooltip (const char * text)
  Sets the current tooltip text.

- uchar damage () const
  Returns non-zero if draw() needs to be called.

- void damage (uchar c)
  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- int damage_resize (int, int, int, int)
  Internal use only.

- void deactivate ()
  Deactivates the widget.

- Fl_Image * deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

- const Fl_Image * deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

- void deimage (Fl_Image & img)
  Sets the image to use as part of the widget label when in the inactive state.

- void deimage (Fl_Image * img)
  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• int h () const
  Gets the widget height.

• virtual void hide ()
  Makes a widget invisible.

• Fl_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.

• const Fl_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.

• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.

• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.

• int image_bound () const
  Returns whether the image is managed by the widget.

• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.

• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().

• const char * label () const
  Gets the current label text.

• void label (const char *text)
  Sets the current label pointer.

• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.

• Fl_Color labelcolor () const
  Gets the label color.

• void labelcolor (Fl_Color c)
  Sets the label color.

• Fl_Font labelfont () const
  Gets the font to use.

• void labelfont (Fl_Font f)
  Sets the font to use.

• Fl_Fontsize labelsize () const
  Gets the font size in pixels.

• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.

• Fl_Labeltype labeltype () const
  Gets the label type.

• void labeltype (Fl_Labeltype a)
  Sets the label type.

• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.

• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.

• void needs_keyboard (bool needs)
Sets whether this widget needs a keyboard.

- unsigned int output () const
  Returns if a widget is used for output only.

- Fl_Group * parent () const
  Returns a pointer to the parent widget.

- void parent (Fl_Group *p)
  Internal use only - "for hacks only".

- void position (int X, int Y)
  Repositions the window or widget.

- void redraw()
  Schedules the drawing of the widget.

- void redraw_label()
  Schedules the drawing of the label.

- virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.

- Fl_Color selection_color () const
  Gets the selection color.

- void selection_color (Fl_Color a)
  Sets the selection color.

- void set_active()
  Marks the widget as active without sending events or changing focus.

- void set_changed()
  Marks the value of the widget as changed.

- void set_output()
  Sets a widget to output only.

- void set_visible()
  Makes the widget visible.

- void set_visible_focus()
  Enables keyboard focus navigation with this widget.

- int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.

- void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

- virtual void show()
  Makes a widget visible.

- void size (int W, int H)
  Changes the size of the widget.

- int take_focus()
  Gives the widget the keyboard focus.

- unsigned int takesevents () const
  Returns if the widget is able to take events.

- int test_shortcut()
  Returns true if the widget's label contains the entered 'x' shortcut.

- const char * tooltip () const
  Gets the current tooltip text.

- void tooltip (const char *text)
  Sets the current tooltip text.

- Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.

- Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data +v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void +v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
• static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
• static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.
Protected Types inherited from Fl_Widget

```
enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 ,
  CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 ,
  CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 ,
  NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 ,
  MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }
```

flags possible values enumeration.

Protected Member Functions inherited from Fl_Dial

```
• void draw () FL_OVERRIDE
  Draws dial at current position and size.
• void draw (int X, int Y , int W, int H)
  Draws dial at given position and size.
• int handle (int event, int X, int Y , int W, int H)
  Allows subclasses to handle event based on given position and size.
```

Protected Member Functions inherited from Fl_Valuator

```
• Fl_Valuator (int X, int Y, int W, int H, const char ∗L)
  Creates a new Fl_Valuator widget using the given position, size, and label string.
• void handle_drag (double newvalue)
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
• void handle_push ()
  Stores the current value in the previous value.
• void handle_release ()
  Called after an FL_WHEN_RELEASE event is received and before the callback.
• int horizontal () const
  Tells if the valuator is an FL_HORIZONTAL one.
• double previous_value () const
  Gets the previous floating point value before an event changed it.
• void set_value (double v)
  Sets the current floating point value.
• double softclamp (double)
  Clamps the value, but accepts v if the previous value is not already out of range.
• virtual void value_damage ()
  Asks for partial redraw.
```

Protected Member Functions inherited from Fl_Widget

```
• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const
```

Generated by Doxygen
Draws the widget box according its box style.

- void **draw_box** (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void **draw_focus** () const
  Draws a focus rectangle around the widget.
- void **draw_focus** (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
- void **draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void **draw_label** () const
  Draws the widget's label at the defined label position.
- void **draw_label** (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- FL_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.
- unsigned int **flags** () const
  Gets the widget flags mask.
- void **h** (int v)
  Internal use only.
- void **set_flag** (unsigned int c)
  Sets a flag in the flags mask.
- void **w** (int v)
  Internal use only.
- void **x** (int v)
  Internal use only.
- void **y** (int v)
  Internal use only.

The documentation for this class was generated from the following files:

- Fl_Line_Dial.H
- Fl_Dial.cxx

### Fl_Mac_App_Menu Class Reference

#### Static Public Member Functions

- static void **custom_application_menu_items** (const Fl_Menu_Item ∗m)
  Adds custom menu item(s) to the application menu of the system menu bar.

#### Static Public Attributes

- static const char ∗**about**
  Localizable text for the "About xxx" application menu item.
- static const char ∗**hide**
  Localizable text for the "Hide xxx" application menu item.
- static const char ∗**hide_others**
  Localizable text for the "Hide Others" application menu item.
- static const char ∗**print**
  Localizable text for the "Print Front Window" application menu item.
- static const char ∗**print_no_titlebar**

Generated by Doxygen
Localizable text for the "Print Front Window" application menu item.

- static const char * quit
  Localizable text for the "Quit xxx" application menu item.
- static const char * services
  Localizable text for the "Services" application menu item.
- static const char * show
  Localizable text for the "Show All" application menu item.
- static const char * toggle_print_titlebar
  Localizable text for the "Toggle print titlebar" application menu item.

### 12.82.1 Member Function Documentation

#### 12.82.1.1 custom_application_menu_items()

static void Fl_Mac_App_Menu::custom_application_menu_items (const Fl_Menu_Item * m) [static]

Adds custom menu item(s) to the application menu of the system menu bar. They are positioned after the "Print Front Window / Toggle printing of titlebar" items, or at their place if they were removed with Fl_Mac_App_Menu::print = "".

**Parameters**

- `m` zero-ending array of Fl_Menu_Item's.

### 12.82.2 Member Data Documentation

#### 12.82.2.1 print

const char* Fl_Mac_App_Menu::print [static]

Localizable text for the "Print Front Window" application menu item.

This menu item and next one won't be displayed if Fl_Mac_App_Menu::print is set to an empty string.

The documentation for this class was generated from the following file:

- mac.H

### 12.83 Fl_Menu_ Class Reference

Base class of all widgets that have a menu in FLTK.

#include <Fl_Menu_.H>

Inheritance diagram for Fl_Menu_:

```
Fl_Widget
  Fl_Menu_
  Fl_Choice  Fl_Menu_Bar  Fl_Menu_Button
  Fl_Scheme_Choice  Fl_Sys_Menu_Bar
```

**Public Member Functions**

- int add (const char *)
This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with \'|\' separating the menu items, and tab separating the menu item names from an optional shortcut string.

- **int add (const char *, int shortcut, Fl_Callback *, void *=0, int=0)**
  Adds a new menu item.

- **int add (const char +a, const char +b, Fl_Callback +c, void +d=0, int e=0)**
  See int Fl_Menu_::add(const char label, int shortcut, Fl_Callback*, void +user_data=0, int flags=0)

- **void clear ()**
  Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.

- **int clear_submenu (int index)**
  Clears the specified submenu pointed to by index of all menu items.

- **void copy (const Fl_Menu_Item *, void *user_data=0)**
  Sets the menu array pointer with a copy of m that will be automatically deleted.

- **Fl_Boxtype down_box () const**
  This box type is used to surround the currently-selected items in the menus.

- **void down_box (Fl_Boxtype b)**
  Sets the box type used to surround the currently-selected items in the menus.

- **Fl_Color down_color () const**
  For back compatibility, same as selection_color()

- **void down_color (unsigned c)**
  For back compatibility, same as selection_color()

- **int find_index (const char *name) const**
  Find the menu item index for a given menu pathname, such as "Edit/Copy".

- **int find_index (const Fl_Menu_Item *item) const**
  Find the index into the menu array for a given item.

- **int find_index (Fl_Callback *cb) const**
  Find the index into the menu array for a given callback cb.

- **const Fl_Menu_Item *find_item (const char *name)**
  Find the menu item for a given menu pathname, such as "Edit/Copy".

- **const Fl_Menu_Item *find_item (Fl_Callback *cb)**
  Find the menu item for the given callback cb.

- **const Fl_Menu_Item *find_item_with_argument (long)**
  Find the menu item for the given user argument v.

- **const Fl_Menu_Item *find_item_with_user_data (void *)**
  Find the menu item for the given user data v.

- **Fl_Menu_ (int, int, int, int, const char *0)**
  Creates a new Fl_Menu_ widget using the given position, size, and label string.

- **void global ()**
  Make the shortcuts for this menu work no matter what window has the focus when you type it.

- **int insert (int index, const char *, int shortcut, Fl_Callback *, void *=0, int=0)**
  Inserts a new menu item at the specified index position.

- **int insert (int index, const char +a, const char +b, Fl_Callback +c, void +d=0, int e=0)**
  See int Fl_Menu_::insert(const char label, int shortcut, Fl_Callback*, void +user_data=0, int flags=0)

- **int item_pathname (char *name, int namelen, const Fl_Menu_Item *finditem=0) const**
  Get the menu 'pathname' for the specified menuitem.

- **const Fl_Menu_Item *menu () const**
  Returns a pointer to the array of Fl_Menu_Items.

- **void menu (const Fl_Menu_Item +m)**
  Sets the menu array pointer directly.

- **Fl_Boxtype menu_box () const**
  Get the box type for the menu popup windows.

- **void menu_box (Fl_Boxtype b)**

Set the box type for the menu popup windows.

- **const Fl_Menu_Item** ∗ menu_end ()
  Finishes menu modifications and returns menu().

- **int** mode (int i) const
  Gets the flags of item i.

- **void** mode (int i, int fl)
  Sets the flags of item i.

- **const Fl_Menu_Item** ∗ mvalue () const
  Returns a pointer to the last menu item that was picked.

- **const Fl_Menu_Item** ∗ picked (const Fl_Menu_Item ∗)
  When user picks a menu item, call this.

- **const Fl_Menu_Item** ∗ prev_mvalue () const
  Returns a pointer to the menu item that was picked before the current one was picked.

- **void** remove (int)
  Deletes item i from the menu.

- **void** replace (int, const char ∗)
  Changes the text of item i.

- **void** setonly (Fl_Menu_Item ∗+item)
  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.

- **void** shortcut (int i, int s)
  Changes the shortcut of item i to s.

- **int** size () const
  This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.

- **void** size (int W, int H)

- **const Fl_Menu_Item** ∗ test_shortcut ()
  Returns the menu item with the entered shortcut (key value).

- **const char** ∗ text () const
  Returns the title of the last item chosen.

- **const char** ∗ text (int i) const
  Returns the title of item i.

- **Fl_Color** textcolor () const
  Get the current color of menu item labels.

- **void** textcolor (Fl_Color c)
  Sets the current color of menu item labels.

- **Fl_Font** textfont () const
  Gets the current font of menu item labels.

- **void** textfont (Fl_Font c)
  Sets the current font of menu item labels.

- **Fl_Fontsize** textsize () const
  Gets the font size of menu item labels.

- **void** textsize (Fl_Fontsize c)
  Sets the font size of menu item labels.

- **int** value () const
  Returns the index into menu() of the last item chosen by the user.

- **int** value (const Fl_Menu_Item ∗)
  The value is the index into menu() of the last item chosen by the user.

- **int** value (int i)
  The value is the index into menu() of the last item chosen by the user.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ as_gl_window () const
- virtual Fl_Group ∗ as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ as_group () const
- virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ as_window () const
- void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
Checks if the widget value changed since the last callback.

- void clear_active()
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed()
  Marks the value of the widget as unchanged.

- void clear_damage(uchar c=0)
  Clears or sets the damage flags.

- void clear_output()
  Sets a widget to accept input.

- void clear_visible()
  Hides the widget.

- void clear_visible_focus()
  Disables keyboard focus navigation with this widget.

- Fl_Color color() const
  Gets the background color of the widget.

- void color(Fl_Color bg)
  Sets the background color of the widget.

- void color(Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

- Fl_Color color2() const
  For back compatibility only.

- void color2(unsigned a)
  For back compatibility only.

- int contains(const Fl_Widget *w) const
  Checks if w is a child of this widget.

- void copy_label(const char *new_label)
  Sets the current label.

- void copy_tooltip(const char *text)
  Sets the current tooltip text.

- uchar damage() const
  Returns non-zero if draw() needs to be called.

- void damage(uchar c)
  Sets the damage bits for the widget.

- void damage(uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- int damage_resize(int, int, int, int)
  Internal use only.

- void deactivate()
  Deactivates the widget.

- Fl_Image *deimage()
  Gets the image that is used as part of the widget label when in the inactive state.

- const Fl_Image *deimage() const
  Gets the image that is used as part of the widget label when in the inactive state.

- void deimage(Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

- void deimage(Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound() const
  Returns whether the inactive image is managed by the widget.

- void do_callback(Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget ∗widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void do_callback (Fl_Widget ∗widget, void ∗arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• virtual void draw ()=0
  Draws the widget.

• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• int h () const
  Gets the widget height.

• virtual int handle (int event)
  Handles the specified event.

• virtual void hide ()
  Makes a widget invisible.

• Fl_Image ∗ image ()
  Gets the image that is used as part of the widget label when in the active state.

• const Fl_Image ∗ image () const
  Gets the image that is used as part of the widget label when in the active state.

• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.

• void image (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the active state.

• int image_bound () const
  Returns whether the image is managed by the widget.

• int inside (const Fl_Widget ∗wgt) const
  Checks if this widget is a child of wgt.

• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().

• const char ∗ label () const
  Gets the current label text.

• void label (const char ∗text)
  Sets the current label pointer.

• void label (Fl_Labeltype a, const char ∗b)
  Shortcut to set the label text and type in one call.

• Fl_Color labelcolor () const
  Gets the label color.

• void labelcolor (Fl_Color c)
  Sets the label color.

• Fl_Font labelfont () const
  Gets the font to use.

• void labelfont (Fl_Font f)
  Sets the font to use.

• Fl_Fontsize labelsize () const
  Gets the font size in pixels.

• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.

• Fl_Labeltype labeltype () const
  Gets the label type.

• void labeltype (Fl_Labeltype a)
  Sets the label type.

• void measure_label (int &ww, int &hh) const
Sets width \( ww \) and height \( hh \) accordingly with the label size.

- **bool needs_keyboard () const**
  
  Returns whether this widget needs a keyboard.

- **void needs_keyboard (bool needs)**

  Sets whether this widget needs a keyboard.

- **unsigned int output () const**

  Returns if a widget is used for output only.

- **Fl_Group *parent () const**

  Returns a pointer to the parent widget.

- **void parent (Fl_Group *p)**

  Internal use only - “for hacks only”.

- **void position (int X, int Y)**

  Repositions the window or widget.

- **void redraw ()**

  Schedules the drawing of the widget.

- **void redraw_label ()**

  Schedules the drawing of the label.

- **virtual void resize (int x, int y, int w, int h)**

  Changes the size or position of the widget.

- **Fl_Color selection_color () const**

  Gets the selection color.

- **void selection_color (Fl_Color a)**

  Sets the selection color.

- **void set_active ()**

  Marks the widget as active without sending events or changing focus.

- **void set_changed ()**

  Marks the value of the widget as changed.

- **void set_output ()**

  Sets a widget to output only.

- **void set_visible ()**

  Makes the widget visible.

- **void set_visible_focus ()**

  Enables keyboard focus navigation with this widget.

- **int shortcut_label () const**

  Returns whether the widget's label uses ‘&’ to indicate shortcuts.

- **void shortcut_label (int value)**

  Sets whether the widget's label uses ‘&’ to indicate shortcuts.

- **virtual void show ()**

  Makes a widget visible.

- **void size (int W, int H)**

  Changes the size of the widget.

- **int take_focus ()**

  Gives the widget the keyboard focus.

- **unsigned int takesevents () const**

  Returns if the widget is able to take events.

- **int test_shortcut ()**

  Returns true if the widget's label contains the entered 'x' shortcut.

- **const char * tooltip () const**

  Gets the current tooltip text.

- **void tooltip (const char *text)**

  Sets the current tooltip text.
- `Fl_Window * top_window () const`
  Returns a pointer to the top-level window for the widget.

- `Fl_Window * top_window_offset (int &xoff, int &yoff) const`
  Finds the x/y offset of the current widget relative to the top-level window.

- `uchar type () const`
  Gets the widget type.

- `void type (uchar t)`
  Sets the widget type.

- `int use_accents_menu ()`
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- `void * user_data () const`
  Gets the user data for this widget.

- `void user_data (Fl_Callback_User_Data *v, bool auto_free)`
  Sets the user data for this widget.

- `void user_data (void *v)`
  Sets the user data for this widget.

- `unsigned int visible () const`
  Returns whether a widget is visible.

- `unsigned int visible_focus () const`
  Checks whether this widget has a visible focus.

- `void visible_focus (int v)`
  Modifies keyboard focus navigation.

- `int visible_r () const`
  Returns whether a widget and all its parents are visible.

- `int w () const`
  Gets the widget width.

- `Fl_When when () const`
  Returns the conditions under which the callback is called.

- `void when (uchar i)`
  Sets the flags used to decide when a callback is called.

- `Fl_Window * window () const`
  Returns a pointer to the nearest parent window up the widget hierarchy.

- `int x () const`
  Gets the widget position in its window.

- `int y () const`
  Gets the widget position in its window.

- `virtual ~Fl_Widget ()`
  Destroys the widget.

Protected Member Functions

- `int item_pathname_ (char *name, int namelen, const Fl_Menu_Item *finditem, const Fl_Menu_Item *menu=0) const`

Protected Member Functions inherited from Fl_Widget

- `void clear_flag (unsigned int c)`
  Clears a flag in the flags mask.

- `void draw_backdrop () const`
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- `void draw_box () const`
  Draws the widget box according its box style.
• **void** `draw_box` (**Fl_Boxtype** t, **Fl_Color** c) const  
  Draws a box of type t, of color c at the widget's position and size.

• **void** `draw_box` (**Fl_Boxtype** t, int x, int y, int w, int h, **Fl_Color** c) const  
  Draws a box of type t, of color c at the position X,Y and size W,H.

• **void** `draw_focus` () const  
  Draws a focus rectangle around the widget.

• **void** `draw_focus` (**Fl_Boxtype** t, int X, int Y, int W, int H) const  
  Draws a focus rectangle around the widget.

• **void** `draw_focus` (**Fl_Boxtype** t, int x, int y, int w, int h, **Fl_Color** bg) const  
  Draws a focus box for the widget at the given position and size.

• **void** `draw_label` () const  
  Draws the widget's label at the defined label position.

• **void** `draw_label` (int, int, int, int) const  
  Draws the label in an arbitrary bounding box.

• **Fl_Widget** (int x, int y, int w, int h, const char ∗=0L)  
  Creates a widget at the given position and size.

• unsigned int `flags` () const  
  Gets the widget flags mask.

• **void** `h` (int v)  
  Internal use only.

• **void** `set_flag` (unsigned int c)  
  Sets a flag in the flags mask.

• **void** `w` (int v)  
  Internal use only.

• **void** `x` (int v)  
  Internal use only.

• **void** `y` (int v)  
  Internal use only.

### Protected Attributes

- uchar `alloc`
- uchar `down_box_`
- **Fl_Boxtype** `menu_box_`
- **Fl_Color** `textcolor_`
- **Fl_Font** `textfont_`
- **Fl_Fontsize** `textsize_`

### Additional Inherited Members

#### Static Public Member Functions inherited from **Fl_Widget**

- static **void** `default_callback` (**Fl_Widget** ∗=widget, void ∗=data)  
  The default callback for all widgets that don't set a callback.

- static unsigned int `label_shortcut` (const char ∗=t)  
  Returns the Unicode value of the 'x' shortcut in a given text.

- static int `test_shortcut` (const char ∗=t, const bool require_alt=false)  
  Returns true if the given text t contains the entered '+x' shortcut.
Protected Types inherited from **Fl_Widget**

```c
enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.
```

### 12.83.1 Detailed Description

Base class of all widgets that have a menu in FLTK.

Currently FLTK provides you with **Fl_Menu_Button**, **Fl_Menu_Bar**, and **Fl_Choice**.

The class contains a pointer to an array of structures of type **Fl_Menu_Item**. The array may either be supplied directly by the user program, or it may be "private": a dynamically allocated array managed by the **Fl_Menu_**.

When the user clicks a menu item, **value()** is set to that item and then:

- If the **Fl_Menu_Item** has a callback set, that callback is invoked with any userdata configured for it. (The **Fl_Menu_** widget's callback is NOT invoked.)
- For any **Fl_Menu_Items** that don't have a callback set, the **Fl_Menu_** widget's callback is invoked with any userdata configured for it. The callback can determine which item was picked using **value()**, **mvalue()**, **item_pathname()**, etc.

The line spacing between menu items can be controlled with the global setting **Fl::menu_linespacing()**.

See also

* **Fl_Widget::shortcut_label(int)**

### 12.83.2 Constructor & Destructor Documentation

#### 12.83.2.1 **Fl_Menu()**

```c
Fl_Menu_::Fl_Menu_ ( int X, int Y, int W, int H, const char ∗ l = 0 )
```

Creates a new **Fl_Menu_** widget using the given position, size, and label string. **menu()** is initialized to null.

### 12.83.3 Member Function Documentation

#### 12.83.3.1 **add()**

```c
int Fl_Menu_::add ( const char ∗ str )
```

This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|' separating the menu items, and tab separating the menu item names from an optional shortcut string. The passed string is split at any '|' characters and then **add(s,0,0,0,0)** is done with each section. This is often useful if you are just using the value, and is compatible with Forms and other GL programs. The section strings use the same special characters as described for the long version of **add()**.

---

Generated by Doxygen
No items must be added to a menu during a callback to the same menu.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str</code></td>
<td>string containing multiple menu labels as described above</td>
</tr>
</tbody>
</table>

Returns

the index into the menu() array, where the entry was added

12.83.3.2 add() [2/2]

```cpp
int Fl_Menu_::add (const char * label, int shortcut, Fl_Callback * callback, void * userdata = 0, int flags = 0)
```

Adds a new menu item.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>label</code></td>
<td>The text label for the menu item.</td>
</tr>
<tr>
<td><code>shortcut</code></td>
<td>Optional keyboard shortcut that can be an int or string: (FL_CTRL+'a') or &quot;^a&quot;. Default 0 if none.</td>
</tr>
<tr>
<td><code>callback</code></td>
<td>Optional callback invoked when user clicks the item. Default 0 if none.</td>
</tr>
<tr>
<td><code>userdata</code></td>
<td>Optional user data passed as an argument to the callback. Default 0 if none.</td>
</tr>
<tr>
<td><code>flags</code></td>
<td>Optional flags that control the type of menu item; see below. Default is 0 for none.</td>
</tr>
</tbody>
</table>

Returns

The index into the menu() array, where the entry was added.

Description

If the menu array was directly set with menu(x), then copy() is done to make a private array.

Since this method can change the internal menu array, any menu item pointers or indices the application may have cached can become stale, and should be recalculated/refreshed.

A menu item's callback must not add() items to its parent menu during the callback.

Due to backwards compatibility and historical restrictions we recommend to use either

- static menu arrays that are not extended during runtime or
- dynamic, extendable menu item arrays that are entirely created by using add() or insert().

This ensures that all menu arrays and strings are copied to internal storage and released when required.

Note

If you create menus from static Fl_Menu_Item arrays and add() or insert() more menu items later, then the menu array is copied to local storage but some local (static) strings may appear to "leak memory". This is a known issue and discouraged usage (see description above) but the impact on memory usage should typically be small.

Detailed Description of Parameters

---

Generated by Doxygen
label

The menu item's label. This argument is required and must not be NULL.

The characters "&", "/", "\", and "_" are treated as special characters in the label string. The "&" character specifies that the following character is an accelerator and will be underlined. The "\" character is used to escape the next character in the string. Labels starting with the "_" character cause a divider to be placed after that menu item.

A label of the form "File/Quit" will create the submenu "File" with a menu item called "Quit".

The label string is copied to new memory and can be freed. The other arguments (including the shortcut) are copied into the menu item unchanged.

If an item exists already with that name then it is replaced with this new one. Otherwise this new one is added to the end of the correct menu or submenu. The return value is the offset into the array that the new entry was placed at.

shortcut

The keyboard shortcut for this menu item.

This parameter is optional, and defaults to 0 to indicate no shortcut.

The shortcut can either be a raw integer value (eg. FL_CTRL+'A') or a string (eg. "^c" or "^97").

Raw integer shortcuts can be a combination of keyboard chars (eg. 'A') and optional keyboard modifiers (see Fl::event_state(), e.g. FL_SHIFT, etc). In addition, FL_COMMAND can be used to denote FL_META under Mac OS X and FL_CTRL under other platforms.

String shortcuts can be specified in one of two ways:

- \<ascii_value> e.g. "97", "^97", "+97", "#97"
- \<ascii_char> e.g. "a", "^a", "+a", "#a"

.. where \<ascii_value> is a decimal value representing an ASCII character (eg. 97 is the ascii code for 'a'), and the optional prefixes enhance the value that follows. Multiple prefixes must appear in the order below.

\# - Alt
+ - Shift
^ - Control

Internally, the text shortcuts are converted to integer values using Fl_old_shortcut(const char*).
callback

The callback to invoke when this menu item is selected.

This parameter is optional, and defaults to 0 for no callback.

userdata

The callback’s ‘user data’ that is passed to the callback.

This parameter is optional, and defaults to 0.

flags

These are bit flags to define what kind of menu item this is.

This parameter is optional, and defaults to 0 to define a ‘regular’ menu item.

These flags can be ‘OR’ed together:

- FL_MENU_INACTIVE // Deactivate menu item (gray out)
- FL_MENU_TOGGLE // Item is a checkbox toggle (shows checkbox for on/off state)
- FL_MENU_VALUE // The on/off state for checkbox/radio buttons (if set, state is ‘on’)
- FL_MENU_RADIO // Item is a radio button (one checkbox of many can be on)
- FL_MENU_INVISIBLE // Item will not show up (shortcut will work)
- FL_SUBMENU_POINTER // Indicates user_data() is a pointer to another menu array
- FL_SUBMENU // This item is a submenu to other items
- FL_MENU_DIVIDER // Creates divider line below this item. Also ends a group of radio buttons.

All other bits in ‘flags’ are reserved and must not be used.

If FL_SUBMENU is set in an item’s flags, then actually two items are added:

- the first item is the menu item (submenu title), as expected, and
- the second item is the submenu terminating item with the label and all other members set to 0.

If you add submenus with the ‘path’ technique, then the corresponding submenu terminators (maybe more than one) are added as well.

Todo Raw integer shortcut needs examples. Dependent on responses to https://www.fltk.org/newsgroups.php?g=ftlk.coredev+v:10086 and results of STR#2344

12.83.3.3 clear()

void Fl_Menu_::clear ( )

Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.

Menus must not be cleared during a callback to the same menu.

12.83.3.4 clear_submenu()

int Fl_Menu_::clear_submenu ( int index )

Clears the specified submenu pointed to by index of all menu items.

This method is useful for clearing a submenu so that it can be re-populated with new items. Example: a "File/Recent Files/..." submenu that shows the last few files that have been opened.

The specified index must point to a submenu.

The submenu is cleared with remove(). If the menu array was directly set with menu(x), then copy() is done to make a private array.
Warning

Since this method can change the internal menu array, any menu item pointers or indices the application may have cached can become stale, and should be recalculated/refreshed.

Example:

```cpp
int index = menubar->find_index("File/Recent"); // get index of "File/Recent" submenu
if ( index != -1 ) menubar->clear_submenu(index); // clear the submenu
menubar->add("File/Recent/Aaa");
mubar->add("File/Recent/Bbb");
```

Parameters

| index | The index of the submenu to be cleared |

Returns

0 on success, -1 if the index is out of range or not a submenu

See also

remove(int)

12.83.3.5 copy()

```cpp
void Fl_Menu_::copy ( const Fl_Menu_Item ∗ m, void ∗ ud = 0 )
```

Sets the menu array pointer with a copy of m that will be automatically deleted. If userdata ud is not NULL, then all user data pointers are changed in the menus as well. See void Fl_Menu_::menu(const Fl_Menu_Item* m).

12.83.3.6 down_box() [1/2]

```cpp
Fl_Boxtype Fl_Menu_::down_box ( ) const [inline]
```

This box type is used to surround the currently-selected items in the menus. If this is FL_NO_BOX then it acts like FL_THIN_UP_BOX and selection_color() acts like FL_WHITE, for back compatibility.

12.83.3.7 down_box() [2/2]

```cpp
void Fl_Menu_::down_box ( Fl_Boxtype b ) [inline]
```

Sets the box type used to surround the currently-selected items in the menus.

12.83.3.8 find_index() [1/3]

```cpp
int Fl_Menu_::find_index ( const char ∗ pathname ) const
```

Find the menu item index for a given menu pathname, such as "Edit/Copy". This method finds a menu item's index position for the given menu pathname, also traversing submenus, but not submenu pointers (FL_SUBMENU_POINTER). To get the menu item pointer for a pathname, use find_item().

Parameters

| in | pathname | The path and name of the menu item to find |

Generated by Doxygen
Returns

The index of the matching item, or -1 if not found.

See also

item_pathname()

12.83.3.9  find_index() [2/3]

int Fl_Menu_::find_index ( const Fl_Menu_Item * item ) const

Find the index into the menu array for a given item. A way to convert a menu item pointer into an index.

A way to convert a menu item pointer into an index.

Does not handle items that are in submenu pointers (FL_SUBMENU_POINTER).

-1 is returned if the item is not in this menu or is part of an FL_SUBMENU_POINTER submenu.

Current implementation is fast and not expensive.

// Convert an index-to-item
int index = 12;
const Fl_Menu_Item *item = mymenu->menu() + index;

// Convert an item-to-index
int index = mymenu->find_index(item);
if ( index == -1 ) { ..error.. }

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>The item to be found</td>
</tr>
</tbody>
</table>

Returns

The index of the item, or -1 if not found.

See also

menu()

12.83.3.10  find_index() [3/3]

int Fl_Menu_::find_index ( Fl_Callback * cb ) const

Find the index into the menu array for a given callback cb.

This method finds a menu item's index position, also traversing submenus, but not submenu pointers (FL_SUBMENU_POINTER). This is useful if an application uses internationalisation and a menu item can not be found using its label. This search is also much faster.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cb</td>
<td>Find the first item with this callback</td>
</tr>
</tbody>
</table>

Returns

The index of the item with the specific callback, or -1 if not found
12.83.3.11 find_item() [1/2]

const Fl_Menu_Item * Fl_Menu_::find_item (const char * pathname)

Find the menu item for a given menu pathname, such as “Edit/Copy”.
This method finds a menu item in the menu array, also traversing submenus, but not submenu pointers (FL_SUBMENU_POINTER).
To get the menu item’s index, use find_index(const char*)

Example:
Fl_Menu_Bar *menubar = new Fl_Menu_Bar(....);
menubar->add("File/&Open");
menubar->add("File/&Save");
menubar->add("Edit/&Copy");
// [...]
Fl_Menu_Item *item;
if ( ( item = (Fl_Menu_Item*)menubar->find_item("File/&Open") ) != NULL ) {
  item->labelcolor(FL_RED);
}
if ( ( item = (Fl_Menu_Item*)menubar->find_item("Edit/&Copy") ) != NULL ) {
  item->labelcolor(FL_GREEN);
}

Parameters

| pathname | The path and name of the menu item |

Returns

The item found, or NULL if not found

See also

find_index(const char*), find_item(Fl_Callback*), item_pathname()

12.83.3.12 find_item() [2/2]

const Fl_Menu_Item * Fl_Menu_::find_item (Fl_Callback * cb)

Find the menu item for the given callback cb.
This method finds a menu item in a menu array, also traversing submenus, but not submenu pointers. This is useful if an application uses internationalisation and a menu item can not be found using its label. This search is also much faster.

Parameters

| in cb | find the first item with this callback |

Returns

The item found, or NULL if not found

See also

find_item(const char*)

12.83.3.13 find_item_with_argument()

const Fl_Menu_Item * Fl_Menu_::find_item_with_argument (long v)

Generated by Doxygen
Find the menu item for the given user argument `v`.

Parameters

```
in  v  find the first item with this user argument
```

Returns

The item found, or NULL if not found

See also

`find_item(const char*)`

### 12.83.3.14 find_item_with_user_data()

```cpp
class Fl_Menu_Item * Fl_Menu_::find_item_with_user_data (void * v)
```

Find the menu item for the given user data `v`.

Parameters

```
in  v  find the first item with this user data
```

Returns

The item found, or NULL if not found

See also

`find_item(const char*)`

### 12.83.3.15 global()

```cpp
void Fl_Menu_::global ( )
```

Make the shortcuts for this menu work no matter what window has the focus when you type it. This is done by using `Fl::add_handler()`. This `Fl_Menu_` widget does not have to be visible (ie the window it is in can be hidden, or it does not have to be put in a window at all). Currently there can be only one `global()` menu. Setting a new one will replace the old one. There is no way to remove the `global()` setting (so don't destroy the widget!)

### 12.83.3.16 insert()

```cpp
int Fl_Menu_::insert (int index,
    const char * label,
    int shortcut,
    Fl_Callback * callback,
    void * userdata = 0,
    int flags = 0)
```

Inserts a new menu item at the specified `index` position. If `index` is -1, the menu item is appended; same behavior as `add()`. To properly insert a menu item, `label` must be the name of the item (eg. “Quit”), and not a ‘menu pathname’ (eg. “File/Quit”). If a menu pathname is specified, the value of `index` is ignored, the new item’s position defined by the pathname.

For more details, see `add()`. Except for the `index` parameter, `add()` has more detailed information on parameters and behavior, and is functionally equivalent.
Parameters

| in | index | The menu array’s index position where the new item is inserted. If -1, behavior is the same as add(). |
| in | label | The text label for the menu item. If the label is a menu pathname, index is ignored, and the pathname indicates the position of the new item. |
| in | shortcut | Optional keyboard shortcut. Can be an int (FL_CTRL+‘a’) or a string (“a”). Default is 0. |
| in | callback | Optional callback invoked when user clicks the item. Default 0 if none. |
| in | userdata | Optional user data passed as an argument to the callback. Default 0 if none. |
| in | flags | Optional flags that control the type of menu item; see add() for more info. Default is 0 for none. |

Returns

The index into the menu() array, where the entry was added.

See also

add()

12.83.3.17 item_pathname()

int Fl_Menu_::item_pathname ( 
    char * name, 
    int namelen, 
    const Fl_Menu_Item * finditem = 0 ) const

Get the menu ‘pathname’ for the specified menuitem. If finditem==NULL, mvalue() is used (the most recently picked menuitem).

Example:

```c
Fl_Menu_Bar *menubar = 0;
void my_menu_callback(Fl_Widget*,void*) {
    char name[80];
    if ( menubar->item_pathname(name, sizeof(name)-1) == 0 ) { // recently picked item
        if ( strcmp(name, "File/&Open") == 0 ) { .. } // open invoked
        if ( strcmp(name, "File/&Save") == 0 ) { .. } // save invoked
        if ( strcmp(name, "Edit/&Copy") == 0 ) { .. } // copy invoked
    }
}
int main() {
    ...
    menubar = new Fl_Menu_Bar(..);
    menubar->add("File/&Open", 0, my_menu_callback);
    menubar->add("File/&Save", 0, my_menu_callback);
    menubar->add("Edit/&Copy", 0, my_menu_callback);
    ...
}
```

Returns

- 0 : OK (name has menuitem’s pathname)
- -1 : item not found (name="")
- -2 : ‘name’ not large enough (name="")

See also

find_item()

12.83.3.18 menu() [1/2]

const Fl_Menu_Item * Fl_Menu_::menu ( ) const [inline]

Returns a pointer to the array of Fl_Menu_Items. This will either be the value passed to menu(value) or the private copy or an internal (temporary) location (see note below).

Generated by Doxygen
Note

_Implementation details - may be changed in the future._ All modifications of the menu array are done by copying the entire menu array to an internal storage for optimization of memory allocations, for instance when using `add()` or `insert()`. While this is done, `menu()` returns the pointer to this internal location. The entire menu will be copied back to private storage when needed, i.e. when another `Fl_Menu_` is modified. You can force this reallocation after you're done with all menu modifications by calling `Fl_Menu_::menu_end()` to make sure `menu()` returns a permanent pointer to private storage (until the menu is modified again). Note also that some menu methods (e.g. `Fl_Menu_Button::popup()`) call `menu_end()` internally to ensure a consistent menu array while the menu is open.

See also

- `size()` – returns the size of the `Fl_Menu_Item` array.
- `menu_end()` – finish menu modifications (optional)

**Example:** How to walk the array:

```c
for ( int t=0; t<menubar->size(); t++ ) { // walk array of items
    const Fl_Menu_Item &item = menubar->menu()[t]; // get each item
    fprintf(stderr, "item #%d -- label=%s, value=%s type=%s\n",
            t, item.label() ? item.label() : "(Null)", // menu terminators have NULL labels
            (item.flags & FL_MENU_VALUE) ? "set" : "clear", // value of toggle or radio items
            (item.flags & FL_SUBMENU) ? "Submenu" : "Item"); // see if item is a submenu or actual item
}
```

### 12.83.3.19 menu()

```c
void Fl_Menu_::menu (const Fl_Menu_Item *m)
```

Set the menu array pointer directly. If the old menu is private it is deleted. NULL is allowed and acts the same as a zero-length menu. If you try to modify the array (with `add()`, `replace()`, or `remove()`) a private copy is automatically done.

### 12.83.3.20 menu_box()

```c
Fl_Boxtype Fl_Menu_::menu_box ( ) const [inline]
```

Get the box type for the menu popup windows.

Returns

the box type, or FL_NO_BOX if `Fl_Menu_::box()` is to be used instead

### 12.83.3.21 menu_box()

```c
void Fl_Menu_::menu_box ( Fl_Boxtype b ) [inline]
```

Set the box type for the menu popup windows. If `menu_box` set to FL_NO_BOX, the menu window will use `Fl_Menu_::box()` instead.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>b</th>
<th>new box type or FL_NO_BOX</th>
</tr>
</thead>
</table>

### 12.83.3.22 menu_end()

```c
const Fl_Menu_Item * Fl_Menu_::menu_end ( )
```

Finishes menu modifications and returns `menu()`.

Call `menu_end()` after using `add()`, `insert()`, `remove()`, or any other methods that may change the menu array if you want to access the menu array anytime later with `menu()`. This should be called only once after the _last_ menu modification for performance reasons.
Does nothing if the menu array is already in a private location.
Some methods like Fl_Menu_Button::popup() call this method before their menu is opened.

Note
After menu changes like add(), insert(), etc. menu() would return a pointer to a temporary internal menu array
that may be relocated at unexpected times. This is due to performance considerations and may be changed
w/o further notice.

Since
1.4.0

Returns
New Fl_Menu_Item array pointer.

See also
Fl_Menu_::menu()

12.83.3.23 mode() [1/2]

int Fl_Menu_::mode (  
    int i ) const [inline]

Gets the flags of item i.
For a list of the flags, see Fl_Menu_Item.

12.83.3.24 mode() [2/2]

void Fl_Menu_::mode (  
    int i,  
    int f) [inline]

Sets the flags of item i.
For a list of the flags, see Fl_Menu_Item.

12.83.3.25 mvalue()

const Fl_Menu_Item ∗ Fl_Menu_::mvalue ( ) const [inline]

Returns a pointer to the last menu item that was picked.

12.83.3.26 picked()

const Fl_Menu_Item ∗ Fl_Menu_::picked (  
    const Fl_Menu_Item ∗ v )

When user picks a menu item, call this.
It will do the callback. Unfortunately this also casts away const for the checkboxes, but this was necessary so
non-checkbox menus can really be declared const...

12.83.3.27 prev_mvalue()

const Fl_Menu_Item ∗ Fl_Menu_::prev_mvalue ( ) const [inline]

Returns a pointer to the menu item that was picked before the current one was picked.
This call gives devs additional details how a user changed a choice in the Fl_Choice widget.
12.83.3.28  remove()

```cpp
void Fl_Menu_::remove ( int i )
```

Deletes item i from the menu.
If the menu array was directly set with menu(x) then copy() is done to make a private array.
No items must be removed from a menu during a callback to the same menu.

**Parameters**

- `i`: index into menu array

12.83.3.29  replace()

```cpp
void Fl_Menu_::replace ( int i, const char * str )
```

Changes the text of item i.
This is the only way to get slash into an add()'ed menu item. If the menu array was directly set with menu(x) then copy() is done to make a private array.

**Parameters**

- `i`: index into menu array
- `str`: new label for menu item at index i

12.83.3.30  size()

```cpp
int Fl_Menu_::size ( ) const
```

This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.
This includes the "terminator" item at the end. To copy a menu array you need to copy size()∗sizeof(Fl_Menu_Item) bytes. If the menu is NULL this returns zero (an empty menu will return 1).

12.83.3.31  test_shortcut()

```cpp
const Fl_Menu_Item * Fl_Menu_::test_shortcut ( ) [inline]
```

Returns the menu item with the entered shortcut (key value).
This searches the complete menu() for a shortcut that matches the entered key value. It must be called for a FL_KEYBOARD or FL_SHORTCUT event.
If a match is found, the menu’s callback will be called.

**Returns**

- matched Fl_Menu_Item or NULL.

12.83.3.32  text [1/2]

```cpp
const char * Fl_Menu_::text ( ) const [inline]
```

Returns the title of the last item chosen.

12.83.3.33  text [2/2]

```cpp
const char * Fl_Menu_::text ( int i ) const [inline]
```

Returns the title of item i.
12.83.3.34  textcolor()

Fl_Color Fl_Menu_::textcolor ( ) const  [inline]
Get the current color of menu item labels.

12.83.3.35  textfont() [1/2]

Fl_Font Fl_Menu_::textfont ( ) const  [inline]
Gets the current font of menu item labels.

12.83.3.36  textfont() [2/2]

void Fl_Menu_::textfont (     Fl_Font c )  [inline]
Sets the current font of menu item labels.

12.83.3.37  textsize() [1/2]

Fl_Fontsize Fl_Menu_::textsize ( ) const  [inline]
Gets the font size of menu item labels.

12.83.3.38  textsize() [2/2]

void Fl_Menu_::textsize (     Fl_Fontsize c )  [inline]
Sets the font size of menu item labels.

12.83.3.39  value() [1/3]

int Fl_Menu_::value ( ) const  [inline]
Returns the index into menu() of the last item chosen by the user.
It is zero initially.

12.83.3.40  value() [2/3]

int Fl_Menu_::value (     const Fl_Menu_Item * m )
The value is the index into menu() of the last item chosen by the user.
It is zero initially. You can set it as an integer, or set it with a pointer to a menu item. The set routines return non-zero
if the new value is different than the old one.

12.83.3.41  value() [3/3]

int Fl_Menu_::value (     int i )  [inline]
The value is the index into menu() of the last item chosen by the user.
It is zero initially. You can set it as an integer, or set it with a pointer to a menu item. The set routines return non-zero if the new value is different than the old one.

The documentation for this class was generated from the following files:

- Fl_Menu_.H
- Fl_Menu_.debug-disabled.cxx
- Fl_Menu_.delete-labels.cxx
- Fl_Menu_.cxx
- Fl_Menu_add.cxx
- Fl_Menu_global.cxx

12.84 Fl_Menu_Bar Class Reference

This widget provides a standard menubar interface.

```
#include <Fl_Menu_Bar.H>
```

Inheritance diagram for Fl_Menu_Bar:

```
Fl_Widget
   \|-- Fl_Menu_
       \|-- Fl_Menu_Bar
           \|-- Fl_Sys_Menu_Bar
```

Public Member Functions

- `Fl_Menu_Bar (int X, int Y, int W, int H, const char ∗l=0)`
  Creates a new Fl_Menu_Bar widget using the given position, size, and label string.

- `int handle (int) FL_OVERRIDE`
  Handles the specified event.

- `virtual void update ()`
  Updates the menu bar after any change to its items.

Public Member Functions inherited from Fl_Menu_

- `int add (const char ∗)`
  This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with ' ' separating the menu items, and tab separating the menu item names from an optional shortcut string.

- `int add (const char ∗, int shortcut, Fl_Callback ∗, void ∗=0, int=0)`
  Adds a new menu item.

- `int add (const char ∗a, const char ∗b, Fl_Callback ∗c, void ∗d=0, int e=0)`
  See int Fl_Menu_::add(const char∗ label, int shortcut, Fl_Callback∗, void ∗user_data=0, int flags=0)

- `void clear ()`
  Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.

- `int clear_submenu (int index)`
  Clears the specified submenu pointed to by index of all menu items.

- `void copy (const Fl_Menu_Item ∗m, void ∗user_data=0)`
  Sets the menu array pointer with a copy of m that will be automatically deleted.

- `Fl_Boxtype down_box () const`
  This box type is used to surround the currently-selected items in the menus.
• void **down_box** (**Fl_Boxtype** b)
  
  Sets the box type used to surround the currently-selected items in the menus.

• **Fl_Color** **down_color** () const
  
  For back compatibility, same as **selection_color**()

• void **down_color**(unsigned c)
  
  For back compatibility, same as **selection_color**()

• int **find_index**(const char ∗name) const
  
  Find the menu item index for a given menu pathname, such as "Edit/Copy".

• int **find_index**(const **Fl_Menu_Item** ∗item) const
  
  Find the index into the menu array for a given item.

• int **find_index**(**Fl_Callback** ∗cb) const
  
  Find the index into the menu array for a given callback cb.

• const **Fl_Menu_Item** ∗**find_item**(const char ∗name)
  
  Find the menu item for a given menu pathname, such as "Edit/Copy".

• const **Fl_Menu_Item** ∗**find_item**(const **Fl_Callback** ∗cb)
  
  Find the menu item for the given callback cb.

• const **Fl_Menu_Item** ∗**find_item_with_argument**(long)
  
  Find the menu item for the given user argument v.

• const **Fl_Menu_Item** ∗**find_item_with_user_data**(void ∗)
  
  Find the menu item for the given user data v.

• **Fl_Menu_**(int, int, int, int, const char ∗=0)
  
  Creates a new **Fl_Menu_** widget using the given position, size, and label string.

• void **global** ()
  
  Make the shortcuts for this menu work no matter what window has the focus when you type it.

• int **insert**(int index, const char ∗, int shortcut, **Fl_Callback** ∗, void ∗=0, int=0)
  
  Inserts a new menu item at the specified index position.

• int **insert**(int index, const char ∗a, const char ∗b, **Fl_Callback** ∗c, void ∗d=0, int e=0)
  
  See int **Fl_Menu_::insert**(const char ∗label, int shortcut, **Fl_Callback** ∗, void ∗user_data=0, int flags=0)

• int **item_pathname**(char ∗name, int namelen, const **Fl_Menu_Item** ∗finditem=0) const
  
  Get the menu 'pathname' for the specified menuitem.

• const **Fl_Menu_Item** ∗**menu** () const
  
  Returns a pointer to the array of **Fl_Menu_Items**.

• void **menu**(const **Fl_Menu_Item** ∗m)
  
  Sets the menu array pointer directly.

• **Fl_Boxtype** **menu_box** () const
  
  Get the box type for the menu popup windows.

• void **menu_box**(**Fl_Boxtype** b)
  
  Set the box type for the menu popup windows.

• const **Fl_Menu_Item** ∗**menu_end** ()
  
  Finishes menu modifications and returns menu()(

• int **mode**(int i) const
  
  Gets the flags of item i.

• void **mode**(int i, int fl)
  
  Sets the flags of item i.

• const **Fl_Menu_Item** ∗**mvalue** () const
  
  Returns a pointer to the last menu item that was picked.

• const **Fl_Menu_Item** ∗**picked**(const **Fl_Menu_Item** ∗)
  
  When user picks a menu item, call this.

• const **Fl_Menu_Item** ∗**prev_mvalue** () const
  
  Returns a pointer to the menu item that was picked before the current one was picked.

• void **remove**(int)
Deletes item \( i \) from the menu.

- **void replace** (int, const char *)
  Changes the text of item \( i \).

- **void setonly** (Fl_Menu_Item *item)
  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.

- **void shortcut** (int i, int s)
  Changes the shortcut of item \( i \) to \( s \).

- **int size () const**
  This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.

- **void size** (int W, int H)

- **const Fl_Menu_Item * test_shortcut ()**
  Returns the menu item with the entered shortcut (key value).

- **const char * text () const**
  Returns the title of the last item chosen.

- **const char * text (int i) const**
  Returns the title of item \( i \).

- **Fl_Color textcolor () const**
  Get the current color of menu item labels.

- **void textcolor** (Fl_Color c)
  Sets the current color of menu item labels.

- **Fl_Font textfont () const**
  Gets the current font of menu item labels.

- **void textfont** (Fl_Font c)
  Sets the current font of menu item labels.

- **Fl_Fontsize textsize () const**
  Gets the font size of menu item labels.

- **void textsize** (Fl_Fontsize c)
  Sets the font size of menu item labels.

- **int value () const**
  Returns the index into menu() of the last item chosen by the user.

- **int value (const Fl_Menu_Item *)**
  The value is the index into menu() of the last item chosen by the user.

- **int value (int i)**
  The value is the index into menu() of the last item chosen by the user.

**Public Member Functions inherited from Fl_Widget**

- **void _clear_fullscreen ()**
- **void _set_fullscreen ()**
- **void activate ()**
  Activates the widget.

- **unsigned int active () const**
  Returns whether the widget is active.

- **int active_r () const**
  Returns whether the widget and all of its parents are active.

- **Fl_Align align () const**
  Gets the label alignment.

- **void align** (Fl_Align alignment)
  Sets the label alignment.

- **long argument () const**
  Gets the current user data (long) argument that is passed to the callback function.
• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
• virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const * as_gl_window () const
• virtual Fl_Group * as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• virtual Fl_Group const * as_group () const
• virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const * as_window () const
• void bind_deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• Fl_Boxtype box () const
  Gets the box type of the widget.
• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
• Fl_Callback_p callback () const
  Gets the current callback function for the widget.
• void callback (Fl_Callback *cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
  Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.
• void copy_label (const char *new_label)
  Sets the current label.
• void copy_tooltip (const char *text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image *image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image *image () const
  Gets the image that is used as part of the widget label when in the active state.
Sets the image to use as part of the widget label when in the active state.

- **void image (Fl_Image *img)**
  
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const**

  Returns whether the image is managed by the widget.

- **int inside (const Fl_Widget *wgt) const**

  Checks if this widget is a child of wgt.

- **int is_label_copied () const**

  Returns whether the current label was assigned with copy_label().

- **const char * label () const**

  Gets the current label text.

- **void label (const char *text)**

  Sets the current label pointer.

- **void label (Fl_Labeltype a, const char *b)**

  Shortcut to set the label text and type in one call.

- **Fl_Color labelcolor () const**

  Gets the label color.

- **void labelcolor (Fl_Color c)**

  Sets the label color.

- **Fl_Font labelfont () const**

  Gets the font to use.

- **void labelfont (Fl_Font f)**

  Sets the font to use.

- **Fl_Fontsize labelsize () const**

  Gets the font size in pixels.

- **void labelsize (Fl_Fontsize pix)**

  Sets the font size in pixels.

- **Fl_Labeltype labeltype () const**

  Gets the label type.

- **void labeltype (Fl_Labeltype a)**

  Sets the label type.

- **void measure_label (int &ww, int &hh) const**

  Sets width ww and height hh accordingly with the label size.

- **bool needs_keyboard () const**

  Returns whether this widget needs a keyboard.

- **void needs_keyboard (bool needs)**

  Sets whether this widget needs a keyboard.

- **unsigned int output () const**

  Returns if a widget is used for output only.

- **Fl_Group * parent () const**

  Returns a pointer to the parent widget.

- **void parent (Fl_Group *p)**

  Internal use only - "for hacks only".

- **void position (int X, int Y)**

  Repositions the window or widget.

- **void redraw ()**

  Schedules the drawing of the widget.

- **void redraw_label ()**

  Schedules the drawing of the label.

- **virtual void resize (int x, int y, int w, int h)**

  Changes the size or position of the widget.
- **Fl_Color selection_color () const**
  Gets the selection color.

- **void selection_color (Fl_Color a)**
  Sets the selection color.

- **void set_active ()**
  Marks the widget as active without sending events or changing focus.

- **void set_changed ()**
  Marks the value of the widget as changed.

- **void set_output ()**
  Sets a widget to output only.

- **void set_visible ()**
  Makes the widget visible.

- **void set_visible_focus ()**
  Enables keyboard focus navigation with this widget.

- **int shortcut_label () const**
  Returns whether the widget's label uses '&' to indicate shortcuts.

- **void shortcut_label (int value)**
  Sets whether the widget's label uses '&' to indicate shortcuts.

- **virtual void show ()**
  Makes a widget visible.

- **void size (int W, int H)**
  Changes the size of the widget.

- **int take_focus ()**
  Gives the widget the keyboard focus.

- **unsigned int takesevents () const**
  Returns if the widget is able to take events.

- **int test_shortcut ()**
  Returns true if the widget's label contains the entered '&x' shortcut.

- **const char * tooltip () const**
  Gets the current tooltip text.

- **void tooltip (const char *text)**
  Sets the current tooltip text.

- **Fl_Window * top_window () const**
  Returns a pointer to the top-level window for the widget.

- **Fl_Window * top_window_offset (int &xoff, int &yoff) const**
  Finds the x/y offset of the current widget relative to the top-level window.

- **uchar type () const**
  Gets the widget type.

- **void type (uchar t)**
  Sets the widget type.

- **int use_accents_menu ()**
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- **void * user_data () const**
  Gets the user data for this widget.

- **void user_data (Fl_Callback_User_Data *v, bool auto_free)**
  Sets the user data for this widget.

- **void user_data (void *v)**
  Sets the user data for this widget.

- **unsigned int visible () const**
  Returns whether a widget is visible.

- **unsigned int visible_focus () const**
  Returns whether a widget has keyboard focus.
Checks whether this widget has a visible focus.

- void **visible_focus** (int v)
  
  Modifies keyboard focus navigation.

- int **visible_r** () const
  
  Returns whether a widget and all its parents are visible.

- int **w** () const
  
  Gets the widget width.

- **Fl_When** when () const
  
  Returns the conditions under which the callback is called.

- void when (uchar i)
  
  Sets the flags used to decide when a callback is called.

- **Fl_Window** ∗ window () const
  
  Returns a pointer to the nearest parent window up the widget hierarchy.

- int **x** () const
  
  Gets the widget position in its window.

- int **y** () const
  
  Gets the widget position in its window.

- virtual ~**Fl_Widget** ()
  
  Destroys the widget.

**Protected Member Functions**

- void draw () FL_OVERRIDE
  
  Draws the widget.

**Protected Member Functions inherited from Fl_Menu_Item**

- int item_pathname_ (char ∗name, int namelen, const Fl_Menu_Item ∗finditem, const Fl_Menu_Item ∗menu=0) const

**Protected Member Functions inherited from Fl_Widget**

- void clear_flag (unsigned int c)
  
  Clears a flag in the flags mask.

- void draw_backdrop () const
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void draw_box () const
  
  Draws the widget box according its box style.

- void draw_box (Fl_Boxtype t, Fl_Color c) const
  
  Draws a box of type t, of color c at the widget's position and size.

- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type t, of color c at the position X,Y and size W,H.

- void draw_focus () const
  
  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.

- void draw_label () const
  
  Draws the widget's label at the defined label position.

- void draw_label (int, int, int) const
  
  Draws the label in an arbitrary bounding box.
• **FL_Widget (int x, int y, int w, int h, const char *label=0L)**
  
  Creates a widget at the given position and size.

• unsigned int **flags () const**
  
  Gets the widget flags mask.

• void **h (int v)**
  
  Internal use only.

• void **set_flag (unsigned int c)**
  
  Sets a flag in the flags mask.

• void **w (int v)**
  
  Internal use only.

• void **x (int v)**
  
  Internal use only.

• void **y (int v)**
  
  Internal use only.

Friends

• class **Fl_Sys_Menu_Bar_Driver**

Additional Inherited Members

Static Public Member Functions inherited from **FL_Widget**

• static void **default_callback (Fl_Widget *widget, void *data)**
  
  The default callback for all widgets that don’t set a callback.

• static unsigned int **label_shortcut (const char *t)**
  
  Returns the Unicode value of the ‘&x’ shortcut in a given text.

• static int **test_shortcut (const char *, const bool require_alt=false)**
  
  Returns true if the given text t contains the entered ‘&x’ shortcut.

Protected Types inherited from **FL_Widget**

• enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,  
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,  
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,  
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,  
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,  
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,  
  AUTO_DELETE_USER_DATA = 1<<23,  
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31}

  flags possible values enumeration.

Protected Attributes inherited from **FL_Menu**

• uchar alloc

• uchar down_box_

• Fl_Boxtype menu_box_

• Fl_Font textfont_

• Fl_Fontsize textsize_
12.84 Fl_Menu_Bar Class Reference

12.84.1 Detailed Description

This widget provides a standard menubar interface. Usually you will put this widget along the top edge of your window. The height of the widget should be 30 for the menu titles to draw correctly with the default font.

The items on the bar and the menus they bring up are defined by a single Fl_Menu_Item array. Because a Fl_Menu_Item array defines a hierarchy, the top level menu defines the items in the menubar, while the submenus define the pull-down menus. Sub-sub menus and lower pop up to the right of the submenus.

![Figure 12.28 menubar](image)

If there is an item in the top menu that is not a title of a submenu, then it acts like a "button" in the menubar. Clicking on it will pick it.

When the user clicks a menu item, value() is set to that item and then:

- The item's callback is done if one has been set; the Fl_Menu_Bar is passed as the Fl_Widget argument, along with any userdata configured for the callback.
- If the item does not have a callback, the Fl_Menu_Bar's callback is done instead, along with any userdata configured for the callback. The callback can determine which item was picked using value(), mvalue(), item_pathname(), etc.

Submenus will also pop up in response to shortcuts indicated by putting a '&' character in the name field of the menu item. If you put a '&' character in a top-level "button" then the shortcut picks it. The '&' character in submenus is ignored until the menu is popped up.

Typing the shortcut() of any of the menu items will cause callbacks exactly the same as when you pick the item with the mouse.

12.84.2 Constructor & Destructor Documentation

12.84.2.1 Fl_Menu_Bar()

Fl_Menu_Bar::Fl_Menu_Bar ( int X, int Y, int W, int H, const char * l = 0 )

Creates a new Fl_Menu_Bar widget using the given position, size, and label string. The default boxtype is FL_UP_BOX.

The constructor sets menu() to NULL. See Fl_Menu_ for the methods to set or change the menu.

labelsiz(), labelfont(), and labelcolor() are used to control how the menubar items are drawn. They are initialized from the Fl_Menu static variables, but you can change them if desired.

label() is ignored unless you change align() to put it outside the menubar.

The destructor removes the Fl_Menu_Bar widget and all of its menu items.

12.84.3 Member Function Documentation

12.84.3.1 draw()

void Fl_Menu_Bar::draw ( ) [protected], [virtual]

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

---

Generated by Doxygen
FL_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()

Implements Fl_Widget.
Reimplemented in Fl_Sys_Menu_Bar.

12.84.3.2 handle()

```cpp
int Fl_Menu_Bar::handle (int event) [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class.
This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

**Parameters**

| in event | the kind of event received |

**Return values**

| 0 | if the event was not used or understood |
| 1 | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Widget.

12.84.3.3 update()

```cpp
virtual void Fl_Menu_Bar::update ( ) [inline], [virtual]
```

Updates the menu bar after any change to its items.
This is useful when the menu bar can be an Fl_Sys_Menu_Bar object.
Reimplemented in Fl_Sys_Menu_Bar.
The documentation for this class was generated from the following files:

- Fl_Menu_Bar.H
- Fl_Menu_Bar.cxx

12.85 Fl_Menu_Button Class Reference

This is a button that when pushed pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects.
```cpp
#include <Fl_Menu_Button.H>
```
Inheritance diagram for Fl_Menu_Button:
Public Types

• enum popup_buttons {
   POPUP1, POPUP2, POPUP12, POPUP3,
   POPUP13, POPUP23, POPUP123 }

   indicate what mouse buttons pop up the menu.

Public Member Functions

• Fl_Menu_Button (int, int, int, int, const char ∗=0)
   Creates a new Fl_Menu_Button widget using the given position, size, and label string.

• int handle (int) FL_OVERRIDE
   Handles the specified event.

• const Fl_Menu_Item ∗ popup ()
   Act exactly as though the user clicked the button or typed the shortcut key.

Public Member Functions inherited from Fl_Menu_

• int add (const char ∗)
   This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|'
   separating the menu items, and tab separating the menu item names from an optional shortcut string.

• int add (const char ∗, int shortcut, Fl_Callback ∗, void ∗=0, int=0)
   Adds a new menu item.

• int add (const char ∗a, const char ∗b, Fl_Callback ∗c, void ∗d=0, int e=0)
   See int Fl_Menu_::add(const char∗ label, int shortcut, Fl_Callback∗, void∗ user_data=0, int flags=0)

• void clear ()
   Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.

• int clear_submenu (int index)
   Clears the specified submenu pointed to by index of all menu items.

• void copy (const Fl_Menu_Item ∗m, void ∗user_data=0)
   Sets the menu array pointer with a copy of m that will be automatically deleted.

• Fl_Boxtype down_box () const
   This box type is used to surround the currently-selected items in the menus.

• void down_box (Fl_Boxtype b)
   Sets the box type used to surround the currently-selected items in the menus.

• Fl_Color down_color () const
   For back compatibility, same as selection_color()

• void down_color (unsigned c)
   For back compatibility, same as selection_color()

• int find_index (const char ∗name) const
   Find the menu item index for a given menu pathname, such as "Edit/Copy".

• int find_index (const Fl_Menu_Item ∗item) const
   Find the index into the menu array for a given item.

• int find_index (Fl_Callback ∗cb) const
   Find the index into the menu array for a given callback cb.
• const Fl_Menu_Item * find_item (const char *name)
  Find the menu item for a given menu pathname, such as "Edit/Copy".

• const Fl_Menu_Item * find_item (Fl_Callback *)
  Find the menu item for the given callback cb.

• const Fl_Menu_Item * find_item_with_argument (long)
  Find the menu item for the given user argument v.

• const Fl_Menu_Item * find_item_with_user_data (void *)
  Find the menu item for the given user data v.

• Fl_Menu_ (int, int, int, int, const char * = 0)
  Creates a new Fl_Menu_ widget using the given position, size, and label string.

• void global ()
  Make the shortcuts for this menu work no matter what window has the focus when you type it.

• int insert (int index, const char *a, int shortcut, Fl_Callback *, void * = 0, int = 0)
  Inserts a new menu item at the specified index position.

• int insert (int index, const char *a, const char *, Fl_Callback *, void * = 0, int = 0)
  See int Fl_Menu_::insert(const char *, int shortcut, Fl_Callback *, void *user_data=0, int flags=0)

• int item_pathname (char *, int namelen, const Fl_Menu_Item *) const
  Get the menu 'pathname' for the specified menuItem.

• const Fl_Menu_Item * menu () const
  Returns a pointer to the array of Fl_Menu_Items.

• void menu (const Fl_Menu_Item *m)
  Sets the menu array pointer directly.

• Fl_Boxtype menu_box () const
  Get the box type for the menu popup windows.

• void menu_box (Fl_Boxtype b)
  Set the box type for the menu popup windows.

• const Fl_Menu_Item * menu_end ()
  Finishes menu modifications and returns menu().

• int mode (int i) const
  Gets the flags of item i.

• void mode (int i, int fl)
  Sets the flags of item i.

• const Fl_Menu_Item * mvalue () const
  Returns a pointer to the last menu item that was picked.

• const Fl_Menu_Item * picked (const Fl_Menu_Item *)
  When user picks a menu item, call this.

• const Fl_Menu_Item * prev_mvalue () const
  Returns a pointer to the menu item that was picked before the current one was picked.

• void remove (int)
  Deletes item i from the menu.

• void replace (int, const char *)
  Changes the text of item i.

• void setonly (Fl_Menu_Item *)
  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.

• void shortcut (int i, int s)
  Changes the shortcut of item i to s.

• int size () const
  This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.

• void size (int W, int H)

• const Fl_Menu_Item * test_shortcut ()
  Returns the menu item with the entered shortcut (key value).
• const char * text () const
  Returns the title of the last item chosen.
• const char * text (int i) const
  Returns the title of item i.
• Fl_Color textcolor () const
  Get the current color of menu item labels.
• void textcolor (Fl_Color c)
  Sets the current color of menu item labels.
• Fl_Font textfont () const
  Gets the current font of menu item labels.
• void textfont (Fl_Font c)
  Sets the current font of menu item labels.
• Fl_Fontsize textsize () const
  Gets the font size of menu item labels.
• void textsize (Fl_Fontsize c)
  Sets the font size of menu item labels.
• int value () const
  Returns the index into menu() of the last item chosen by the user.
• int value (const Fl_Menu_Item *)
  The value is the index into menu() of the last item chosen by the user.
• int value (int i)
  The value is the index into menu() of the last item chosen by the user.

Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  Activates the widget.
• unsigned int active () const
  Returns whether the widget is active.
• int active_r () const
  Returns whether the widget and all of its parents are active.
• Fl_Align align () const
  Gets the label alignment.
• void align (Fl_Align alignment)
  Sets the label alignment.
• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
• virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const * as_gl_window () const
• virtual Fl_Group * as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• virtual Fl_Group const * as_group () const
• virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const * as_window () const
• void bind_deimage (Fl_Image *img)
Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image (Fl_Imag* img)
  Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- Fl_Boxtype box () const
  Gets the box type of the widget.

- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.

- Fl_Callback_p callback () const
  Gets the current callback function for the widget.

- void callback (Fl_Callback *cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.

- void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.

- unsigned int changed () const
  Checks if the widget value changed since the last callback.

- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  Clears or sets the damage flags.

- void clear_output ()
  Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.

- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

- Fl_Color color () const
  Gets the background color of the widget.

- void color (Fl_Color bg)
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

- Fl_Color color2 () const
  For back compatibility only.

- void contains (unsigned a)
  For back compatibility only.

- int contains (const FL_Widget *w) const
  Checks if w is a child of this widget.

- void copy_label (const char *new_label)
  Sets the current label.
• void **copy_tooltip** (const char *text)
  
  *Sets the current tooltip text.*

• uchar **damage** () const
  
  *Returns non-zero if draw() needs to be called.*

• void **damage** (uchar c)
  
  *Sets the damage bits for the widget.*

• void **damage** (uchar c, int x, int y, int w, int h)
  
  *Sets the damage bits for an area inside the widget.*

• int **damage_resize** (int, int, int)
  
  *Internal use only.*

• void **deactivate** ()
  
  *Deactivates the widget.*

• Fl_IMAGE * **deimage** ()
  
  *Gets the image that is used as part of the widget label when in the inactive state.*

• const Fl_IMAGE * **deimage** () const
  
  *Gets the image that is used as part of the widget label when in the inactive state.*

• void **deimage** (Fl_IMAGE &img)
  
  *Sets the image to use as part of the widget label when in the inactive state.*

• void **deimage** (Fl_IMAGE * img)
  
  *Sets the image to use as part of the widget label when in the inactive state.*

• int **deimage_bound** () const
  
  *Returns whether the inactive image is managed by the widget.*

• void **do_callback** (Fl_Callback_REASON reason=FL_REASON_UNKNOWN)
  
  *Calls the widget callback function with default arguments.*

• void **do_callback** (Fl_Widget *widget, long arg, Fl_Callback_REASON reason=FL_REASON_UNKNOWN)
  
  *Calls the widget callback function with arbitrary arguments.*

• void **do_callback** (Fl_Widget *widget, void *arg=0, Fl_Callback_REASON reason=FL_REASON_UNKNOWN)
  
  *Calls the widget callback function with arbitrary arguments.*

• void **draw_label** (int, int, int, int, Fl_ALIGN) const
  
  *Draws the label in an arbitrary bounding box with an arbitrary alignment.*

• int **h** () const
  
  *Gets the widget height.*

• virtual void **hide** ()
  
  *Makes a widget invisible.*

• Fl_IMAGE * **image** ()
  
  *Gets the image that is used as part of the widget label when in the active state.*

• const Fl_IMAGE * **image** () const
  
  *Gets the image that is used as part of the widget label when in the active state.*

• void **image** (Fl_IMAGE &img)
  
  *Sets the image to use as part of the widget label when in the active state.*

• void **image** (Fl_IMAGE * img)
  
  *Sets the image to use as part of the widget label when in the active state.*

• int **image_bound** () const
  
  *Returns whether the image is managed by the widget.*

• int **inside** (const Fl_Widget *wgt) const
  
  *Checks if this widget is a child of wgt.*

• int **is_label_copied** () const
  
  *Returns whether the current label was assigned with copy_label().*

• const char * **label** () const
  
  *Gets the current label text.*

• void **label** (const char *text)
Sets the current label pointer.

- void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.

- Fl_Color labelcolor () const
  Gets the label color.

- void labelcolor (Fl_Color c)
  Sets the label color.

- Fl_Font labelfont () const
  Gets the font to use.

- void labelfont (Fl_Font f)
  Sets the font to use.

- Fl_Fontsize labelsize () const
  Gets the font size in pixels.

- void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.

- Fl_Labeltype labeltype () const
  Gets the label type.

- void labeltype (Fl_Labeltype a)
  Sets the label type.

- void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.

- bool needs_keyboard () const
  Returns whether this widget needs a keyboard.

- void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.

- unsigned int output () const
  Returns if a widget is used for output only.

- Fl_Group * parent () const
  Returns a pointer to the parent widget.

- void parent (Fl_Group *p)
  Internal use only - "for hacks only".

- void position (int X, int Y)
  Repositions the window or widget.

- void redraw ()
  Schedules the drawing of the widget.

- void redraw_label ()
  Schedules the drawing of the label.

- virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.

- Fl_Color selection_color () const
  Gets the selection color.

- void selection_color (Fl_Color a)
  Sets the selection color.

- void set_active ()
  Marks the widget as active without sending events or changing focus.

- void set_changed ()
  Marks the value of the widget as changed.

- void set_output ()
  Sets a widget to output only.

- void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int testShortcut ()
  Returns true if the widget's label contains the entered 'x' shortcut.
• const char ∗ tooltip () const
  Gets the current tooltip text.
• void tooltip (const char ∗text)
  Sets the current tooltip text.
• Fl_Window ∗ top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void ∗ user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void ∗v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
Returns a pointer to the nearest parent window up the widget hierarchy.

- **int x () const**
  
  Gets the widget position in its window.

- **int y () const**
  
  Gets the widget position in its window.

- **virtual ~Fl_Widget ()**
  
  Destroys the widget.

Protected Member Functions

- **void draw () FL_OVERRIDE**
  
  Draws the widget.

Protected Member Functions inherited from Fl_Menu_

- **int item_pathname_ (char *name, int namelen, const Fl_Menu_Item *finditem, const Fl_Menu_Item *menu=0) const**

Protected Member Functions inherited from Fl_Widget

- **void clear_flag (unsigned int c)**
  
  Clears a flag in the flags mask.

- **void draw_backdrop () const**
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- **void draw_box () const**
  
  Draws the widget box according its box style.

- **void draw_box (Fl_Boxtype t, Fl_Color c) const**
  
  Draws a box of type t, of color c at the widget's position and size.

- **void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const**
  
  Draws a box of type t, of color c at the position X,Y and size W,H.

- **void draw_focus () const**
  
  Draws a focus rectangle around the widget.

- **void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const**
  
  Draws a focus rectangle around the widget.

- **void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const**
  
  Draws a focus box for the widget at the given position and size.

- **void draw_label () const**
  
  Draws the widget's label at the defined label position.

- **void draw_label (int, int, int, int) const**
  
  Draws the label in an arbitrary bounding box.

- **Fl_Widget (int x, int y, int w, int h, const char *label=0L)**
  
  Creates a widget at the given position and size.

- **unsigned int flags () const**
  
  Gets the widget flags mask.

- **void h (int v)**
  
  Internal use only.

- **void set_flag (unsigned int c)**
  
  Sets a flag in the flags mask.

- **void w (int v)**
  
  Internal use only.

- **void x (int v)**
  
  Internal use only.

- **void y (int v)**
  
  Internal use only.
Static Protected Attributes

- static Fl_Menu_Button* pressed_menu_button_ = NULL

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget* widget, void* data)
  The default callback for all widgets that don’t set a callback.
- static unsigned int label_shortcut (const char* t)
  Returns the Unicode value of the ‘&x’ shortcut in a given text.
- static int test_shortcut (const char* t, const bool require_alt=false)
  Returns true if the given text t contains the entered ‘&x’ shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLtip_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
flags possible values enumeration.

Protected Attributes inherited from Fl_Menu_

- uchar alloc
- uchar down_box_
- Fl_Boxtype menu_box_
- Fl_Color textcolor_
- Fl_Font textfont_
- Fl_Fontsize textsize_

12.85.1 Detailed Description

This is a button that when pushed pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects.
Normally any mouse button will pop up a menu and it is lined up below the button as shown in the picture. However an Fl_Menu_Button may also control a pop-up menu. This is done by setting the type(). If type() is zero a normal menu button is produced. If it is nonzero then this is a pop-up menu. The bits in type() indicate what mouse buttons pop up the menu (see Fl_Menu_Button::popup_buttons).

The menu will also pop up in response to shortcuts indicated by putting a ‘&’ character in the label(). Typing the shortcut() of any of the menu items will cause callbacks exactly the same as when you pick the item with the mouse. The ‘&’ character in menu item names are only looked at when the menu is popped up, however.

When the user clicks a menu item, value() is set to that item and then:

- The item’s callback is done if one has been set; the Fl_Menu_Button is passed as the Fl_Widget+ argument, along with any userdata configured for the callback.
- If the item does not have a callback, the Fl_Menu_Button’s callback is done instead, along with any userdata configured for it. The callback can determine which item was picked using value(), mvalue(), item_pathname(), etc.

### 12.85.2 Member Enumeration Documentation

#### 12.85.2.1 popup_buttons

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POPUP1</td>
<td>pops up with the mouse 1st button.</td>
</tr>
<tr>
<td>POPUP2</td>
<td>pops up with the mouse 2nd button.</td>
</tr>
<tr>
<td>POPUP12</td>
<td>pops up with the mouse 1st or 2nd buttons.</td>
</tr>
<tr>
<td>POPUP3</td>
<td>pops up with the mouse 3rd button.</td>
</tr>
<tr>
<td>POPUP13</td>
<td>pops up with the mouse 1st or 3rd buttons.</td>
</tr>
<tr>
<td>POPUP23</td>
<td>pops up with the mouse 2nd or 3rd buttons.</td>
</tr>
<tr>
<td>POPUP123</td>
<td>pops up with any mouse button.</td>
</tr>
</tbody>
</table>

Values for type() used to indicate what mouse buttons pop up the menu. Fl_Menu_Button::POPUP3 is usually what you want.

Generated by Doxygen
12.85.3 Constructor & Destructor Documentation

12.85.3.1 Fl_Menu_Button()

Fl_Menu_Button::Fl_Menu_Button (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char ∗ l = 0 )

Creates a new Fl_Menu_Button widget using the given position, size, and label string. The default boxtype is FL_UP_BOX. The constructor sets menu() to NULL. See Fl_Menu_ for the methods to set or change the menu.

12.85.4 Member Function Documentation

12.85.4.1 draw()

void Fl_Menu_Button::draw ( ) [protected], [virtual]

Draws the widget. Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead. Override this function to draw your own widgets. If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

Fl_Widget ∗s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()

Implements Fl_Widget.

12.85.4.2 handle()

int Fl_Menu_Button::handle (  
    int event ) [virtual]

Handles the specified event. You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval. One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class. In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

- **event**: the kind of event received

Return values

- 0 if the event was not used or understood
- 1 if the event was used and can be deleted

See also

- Fl_Event

Reimplemented from Fl_Widget.
12.85.4.3 popup()

const Fl_Menu_Item * Fl_Menu_Button::popup ( )

Act exactly as though the user clicked the button or typed the shortcut key.
The menu appears, it waits for the user to pick an item, and if they pick one it sets value() and does the callback or sets changed() as described above. The menu item is returned or NULL if the user dismisses the menu.

Note

Since FLTK 1.4.0 Fl_Menu_::menu_end() is called before the menu pops up to make sure the menu array is located in private storage.

See also

Fl_Menu_::menu_end()

The documentation for this class was generated from the following files:

- Fl_Menu_Button.H
- Fl_Menu_Button.cxx

12.86 Fl_Menu_Item Struct Reference

The Fl_Menu_Item structure defines a single menu item that is used by the Fl_Menu_ class.

#include <Fl_Menu_Item.H>

Public Member Functions

- void activate ()
  Allows a menu item to be picked.
- int active () const
  Gets whether or not the item can be picked.
- int activevisible () const
  Returns non 0 if FL_INACTIVE and FL_INVISIBLE are cleared, 0 otherwise.
- int add (const char *, int shortcut, Fl_Callback *, void *, int=0)
  Adds a menu item.
- int add (const char *, const char *, Fl_Callback *, void *, int e=0)
  See int add(const char *, int shortcut, Fl_Callback *, void *, int)
- long argument () const
  Gets the user_data() argument that is sent to the callback function.
- void argument (long v)
  Sets the user_data() argument that is sent to the callback function.
- Fl_Callback_p callback () const
  Returns the callback function that is set for the menu item.
- void callback (Fl_Callback *c)
  Sets the menu item’s callback function.
- void callback (Fl_Callback *c, void *p)
  Sets the menu item’s callback function and userdata() argument.
- void callback (Fl_Callback0 *c)
  Sets the menu item’s callback function.
- void callback (Fl_Callback1 *c, long p=0)
  Sets the menu item’s callback function and userdata() argument.
- void check ()
  Back compatibility only.
- int checkbox () const
  Returns true if a checkbox will be drawn next to this item.
• int **checked () const**
  Back compatibility only.
• void **clear ()**
  Turns the check or radio item "off" for the menu item.
• void **deactivate ()**
  Prevents a menu item from being picked.
• void **do_callback (Fl_Widget∗ o) const**
  Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.
• void **do_callback (Fl_Widget∗ o, long arg) const**
  Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.
• void **do_callback (Fl_Widget∗ o, void∗ arg) const**
  Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.
• void **draw (int x, int y, int w, int h, const Fl_Menu_Item∗, int t=0) const**
  Draws the menu item in bounding box x,y,w,h, optionally selects the item.
• const Fl_Menu_Item∗ **find_shortcut (int∗ ip=0, const bool require_alt=false) const**
  Search only the top level menu for a shortcut.
• Fl_Menu_Item∗ **first ()**
  Returns the first menu item, same as next(0).
• const Fl_Menu_Item∗ **first () const**
  Returns the first menu item, same as next(0).
• void **hide ()**
  Hides an item in the menu.
• void **image (Fl_Image &image)**
  Compatibility API for FLUID, same as image.label(this).
• void **image (Fl_Image∗ image)**
  Compatibility API for FLUID, same as image->label(this).
• void **image_label (const Fl_Image∗ image)**
  Sets the title (label()) to an icon or image.
• int **insert (int, const char∗, int, Fl_Callback∗, void∗ =0, int t=0)**
  Inserts an item at position index.
• const char∗ **label () const**
  Returns the title (label) of the menu item.
• void **label (const char∗ a)**
  Sets the title (label) of the menu item.
• void **label (Fl_Labeltype a, const char∗ b)**
  Sets the title (label) and the label type of the menu item.
• Fl_Color **labelcolor () const**
  Gets the menu item's label color.
• void **labelcolor (Fl_Color a)**
  Sets the menu item's label color.
• Fl_Font **labelfont () const**
  Gets the menu item's label font.
• void **labelfont (Fl_Font a)**
  Sets the menu item's label font.
• Fl_Fontsize **labelsizesize () const**
  Gets the label font pixel size/height.
• void **labelsizesize (Fl_Fontsize a)**
  Sets the label font pixel size/height.
• Fl_Labeltype **labeltype () const**
  Returns the menu item's labeltype.
• void **labeltype (Fl_Labeltype a)**
Sets the menu item's labeltype.

- **int measure (int h, const Fl_Menu_Item *) const**
  Measures width of label, including effect of & characters.

- **void multi_label (const Fl_Multi_Label *ml)**
  Sets the title (label()) and labeltype() to an Fl_Multi_Label.

- **Fl_Menu_Item * next (int i=1)**
  Advances a pointer by n items through a menu array, skipping the contents of submenus and invisible items.

- **const Fl_Menu_Item * next (int=1) const**
  Advance a pointer by n items through a menu array, skipping the contents of submenus and invisible items.

- **const Fl_Menu_Item * popup (int X, int Y, const char *title=0, const Fl_Menu_Item *picked=0, const Fl_Menu_Item *title=0, const Fl_Menu_Item *menubar=0) const**
  This method is called by widgets that want to display menus.

- **const Fl_Menu_Item * pulldown (int X, int Y, int W, int H, const Fl_Menu_Item *picked=0, const Fl_Menu_Item *title=0, int menubar=0) const**
  Pulldown() is similar to popup(), but a rectangle is provided to position the menu.

- **int radio () const**
  Returns true if this item is a radio item.

- **void set ()**
  Turns the check or radio item "on" for the menu item.

- **void setonly (Fl_Menu_Item const *first=NULL)**
  Turns the radio item "on" for the menu item and turns "off" adjacent radio items set.

- **int shortcut () const**
  Gets what key combination shortcut will trigger the menu item.

- **void shortcut (int s)**
  Sets exactly what key combination will trigger the menu item.

- **void show ()**
  Makes an item visible in the menu.

- **int size () const**
  Size of the menu starting from this menu item.

- **int submenu () const**
  Returns true if either FL_SUBMENU or FL_SUBMENU_POINTER is on in the flags.

- **const Fl_Menu_Item * test_shortcut () const**
  This is designed to be called by a widgets handle() method in response to a FL_SHORTCUT event.

- **void uncheck ()**
  Back compatibility only.

- **void * user_data () const**
  Gets the user_data() argument that is sent to the callback function.

- **void user_data (void *v)**
  Sets the user_data() argument that is sent to the callback function.

- **int value () const**
  Returns the current value of the check or radio item.

- **void value (int v)**
  Sets the current value of the check or radio item.

- **int visible () const**
  Gets the visibility of an item.
Public Attributes

- **Fl_Callback * callback_
  
  menu item callback

- **int flags
  
  menu item flags like FL_MENU_TOGGLE, FL_MENU_RADIO

- **Fl_Color labelcolor_
  
  menu item text color

- **Fl_Font labelfont_
  
  which font for this menu item text

- **Fl_Fontsize labelsize_
  
  size of menu item text

- **uchar labeltype_
  
  how the menu item text looks like

- **int shortcut_
  
  menu item shortcut

- **const char * text
  
  menu item text, returned by label()

- **void * user_data_
  
  menu item user_data for the menu's callback

12.86.1 Detailed Description

The Fl_Menu_Item structure defines a single menu item that is used by the Fl_Menu_ class.

```c
struct Fl_Menu_Item {
    const char* text; // label()
    int shortcut_;    // Item is a checkbox toggle (shows checkbox for on/off state)
    Fl_Callback* callback_; // The on/off state for checkbox/radio buttons (if set, state is 'on')
    Fl_Font labelfont_; // Item is a radio button (one checkbox of many can be on)
    Fl_Fontsize labelsize_; // Item will not show up (shortcut will work)
    Fl_Callback* submenu_; // Indicates user_data() is a pointer to another menu array
    Fl_Menu_Item* context_; // This item is a submenu to other items
    Fl_Menu_Item* parent_;  // Creates divider line below this item. Also ends a group of radio
    Fl_Menu_Item* sibling_; // buttons.
    Fl_Menu_Item* anchor_;  // FL_MENU_HORIZONTAL = 0x100, // ??? -- reserved, internal (do not use)
    Fl_Menu_Item* next_;    // FL_MENU_RESERVED = 0xffffffff // These bits are reserved for internal or future usage (do not use)
    Fl_Menu_Item* previous_;
}
```

Typically menu items are statically defined; for example:

```c
Fl_Menu_Item popup[] = {
    {"alpha", FL_ALT+'a', the_cb, (void*)1},
    {"beta", FL_ALT+'b', the_cb, (void*)2},
    {"gamma", FL_ALT+'c', the_cb, (void*)3, FL_MENU_DIVIDER},
    {"strange", 0, strange_cb},
    {"charm", 0, charm_cb},
    {"truth", 0, truth_cb},
    {"beauty", 0, beauty_cb},
    {"submenu", 0, 0, 0, FL_SUBMENU},
    {"one"},
    {"two"},
    {"three"},
    {0},
    {"inactive", FL_ALT+'i', 0, 0, FL_MENU_INACTIVE|FL_MENU_DIVIDER},
    {"invisible", FL_ALT+'i', 0, 0, FL_MENU_INVISIBLE},
    {"check", FL_ALT+'c', 0, 0, FL_MENU_TOGGLE|FL_MENU_VALUE},
    {"box", FL_ALT+'b', 0, 0, FL_MENU_TOGGLE},
    {0}};
```
A submenu title is identified by the bit FL_SUBMENU in the flags field, and ends with a label() that is NULL. You can nest menus to any depth. A pointer to the first item in the submenu can be treated as an Fl_Menu array itself. It is also possible to make separate submenu arrays with FL_SUBMENU_POINTER flags.

You should use the method functions to access structure members and not access them directly to avoid compatibility problems with future releases of FLTK.

Note

Adding menu items with insert(), add(), or any of its overloaded variants copies the entire menu to internal storage. Using the memory of a static menu array after that would access unused (but not released) memory and thus have no effect.

12.86.2 Member Function Documentation

12.86.2.1 add()

```
int Fl_Menu_Item::add (const char * mytext,
                      int sc,
                      Fl_Callback * cb,
                      void * data = 0,
                      int myflags = 0 )
```

Adds a menu item.

The text is split at '/' characters to automatically produce submenus (actually a totally unnecessary feature as you can now add submenu titles directly by setting FL_SUBMENU in the flags).

Returns

the index into the menu() array, where the entry was added
See also

Fl_Menu_Item::insert(int, const char*, int, Fl_Callback*, void*, int)

12.86.2.2 argument() [1/2]

long Fl_Menu_Item::argument() const [inline]

Gets the user_data() argument that is sent to the callback function.
For convenience you can also define the callback as taking a long argument. This method casts the stored user-
data() argument to long and returns it as a long value.

12.86.2.3 argument() [2/2]

void Fl_Menu_Item::argument (long v) [inline]

Sets the user_data() argument that is sent to the callback function.
For convenience you can also define the callback as taking a long argument. This method casts the given argument
v to void* and stores it in the menu item's userdata() member. This may not be portable to some machines.

12.86.2.4 callback() [1/5]

Fl_Callback_p Fl_Menu_Item::callback() const [inline]

Returns the callback function that is set for the menu item.
Each item has space for a callback function and an argument for that function. Due to back compatibility, the
Fl_Menu_Item itself is not passed to the callback, instead you have to get it by calling ((Fl_Menu_*w)->mvalue()
where w is the widget argument.

12.86.2.5 callback() [2/5]

void Fl_Menu_Item::callback (Fl_Callback * c) [inline]

Sets the menu item's callback function.
This method does not set the userdata() argument.
See also

Fl_Callback_p Fl_MenuItem::callback() const

12.86.2.6 callback() [3/5]

void Fl_Menu_Item::callback (Fl_Callback * c, void * p) [inline]

Sets the menu item's callback function and userdata() argument.
See also

Fl_Callback_p Fl_MenuItem::callback() const

12.86.2.7 callback() [4/5]

void Fl_Menu_Item::callback (Fl_Callback0 * c) [inline]

Sets the menu item's callback function.
This method does not set the userdata() argument.
See also

Fl_Callback_p Fl_MenuItem::callback() const
12.86.2.8 callback() [5/5]

void Fl_Menu_Item::callback (  
    Fl_Callback1 ∗ c,  
    long p = 0 ) [inline]
Sets the menu item's callback function and userdata() argument.  
The argument is cast to void* and stored as the userdata() for the menu item's callback function.

See also
    Fl_Callback_p Fl_MenuItem::callback() const

12.86.2.9 check()

void Fl_Menu_Item::check ( ) [inline]
Back compatibility only.

Deprecated Please use Fl_Menu_Item::set() instead. This method will be removed in FLTK 1.5.0 or later.

See also
    set()

12.86.2.10 checkbox()

int Fl_Menu_Item::checkbox ( ) const [inline]
Returns true if a checkbox will be drawn next to this item.  
This is true if FL_MENU_TOGGLE or FL_MENU_RADIO is set in the flags.

12.86.2.11 checked()

int Fl_Menu_Item::checked ( ) const [inline]
Back compatibility only.

Deprecated Please use Fl_Menu_Item::value() instead. This method will be removed in FLTK 1.5.0 or later.

See also
    value()

12.86.2.12 deactivate()

void Fl_Menu_Item::deactivate ( ) [inline]
Prevents a menu item from being picked.
Note that this will also cause the menu item to appear grayed-out.

12.86.2.13 do_callback() [1/3]

void Fl_Menu_Item::do_callback (  
    Fl_Widget ∗ o ) const [inline]
Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.  
The callback is called with the stored user_data() as its second argument. You must first check that callback() is non-zero before calling this.

12.86.2.14 do_callback() [2/3]

void Fl_Menu_Item::do_callback (  
    Fl_Widget ∗ o,  
    long arg ) const [inline]
Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.  
This call overrides the callback's second argument with the given value arg. long arg is cast to void* when calling the callback. You must first check that callback() is non-zero before calling this.
12.86.2.15 do_callback() [3/3]

```c
void Fl_Menu_Item::do_callback ( 
    Fl_Widget * o, 
    void * arg ) const [inline]
```

Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument. This call overrides the callback's second argument with the given value arg. You must first check that callback() is non-zero before calling this.

12.86.2.16 find_shortcut() const

```c
const Fl_Menu_Item * Fl_Menu_Item::find_shortcut ( 
    int * ip = 0, 
    const bool require_alt = false ) const
```

Search only the top level menu for a shortcut. Either &x in the label or the shortcut fields are used. This tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value.

**Parameters**

| ip | returns the index of the item, if ip is not NULL. |
| require_alt | if true: match only if Alt key is pressed. |

**Returns**

found Fl_Menu_Item or NULL

12.86.2.17 image() [1/2]

```c
void Fl_Menu_Item::image ( 
    Fl_Image & image ) [inline]
```

Compatibility API for FLUID, same as image.label(this).

**Note**

This method is intended for internal use by fluid and may not do what you expect.

12.86.2.18 image() [2/2]

```c
void Fl_Menu_Item::image ( 
    Fl_Image * image ) [inline]
```

Compatibility API for FLUID, same as image->label(this).

**Note**

This method is intended for internal use by fluid and may not do what you expect.

12.86.2.19 image_label()

```c
void Fl_Menu_Item::image_label ( 
    const Fl_Image * image ) [inline]
```

Sets the title (label()) to an icon or image. This sets the labeltype() to _FL_IMAGE_LABEL (note the leading underscore).

**See also**

const char* Fl_Menu_Item::label() const

**Since**

1.4.0
12.86.2.20  insert()

```cpp
int Fl_Menu_Item::insert (  
  int index,  
  const char * mytext,  
  int sc,  
  Fl_Callback * cb,  
  void * data = 0,  
  int myflags = 0)
```

Inserts an item at position `index`. If `index` is -1, the item is added the same way as `Fl_Menu_Item::add()`. If 'mytext' contains any un-escaped front slashes (/), it's assumed a menu pathname is being specified, and the value of `index` will be ignored. In all other aspects, the behavior of `insert()` is the same as `add()`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>index</code></td>
<td>insert new items here</td>
</tr>
<tr>
<td><code>mytext</code></td>
<td>new label string, details see above</td>
</tr>
<tr>
<td><code>sc</code></td>
<td>keyboard shortcut for new item</td>
</tr>
<tr>
<td><code>cb</code></td>
<td>callback function for new item</td>
</tr>
<tr>
<td><code>data</code></td>
<td>user data for new item</td>
</tr>
<tr>
<td><code>myflags</code></td>
<td>menu flags as described in <code>Fl_Menu_Item</code></td>
</tr>
</tbody>
</table>

**Returns**

the index into the menu() array, where the entry was added

12.86.2.21  label() [1/3]

```cpp
const char * Fl_Menu_Item::label ( ) const [inline]
```

Returns the title (label) of the menu item. A NULL here indicates the end of the menu (or of a submenu). A '&' in the item will print an underscore under the next letter, and if the menu is popped up that letter will be a "shortcut" to pick that item. To get a real '&' put two in a row.

The returned value depends on the `labeltype()` that has been used to store the label.

**Returns**

A pointer to the label cast to (const char *)

**Return values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>A pointer to text (const char *)</td>
</tr>
<tr>
<td>(b)</td>
<td>A pointer to an image (Fl_Image *)</td>
</tr>
<tr>
<td>(c)</td>
<td>A pointer to a &quot;multi label&quot; (Fl_Multi_Label *)</td>
</tr>
<tr>
<td><code>NULL</code></td>
<td>End of menu or submenu</td>
</tr>
</tbody>
</table>

**See also**

- `Fl_Menu_Item::label(const char *)`
- `Fl_Menu_Item::label(Fl_Labeltype, const char *)`
- `Fl_Menu_Item::image_label(const Fl_Image *)`
- `Fl_Menu_Item::multi_label(const Fl_Multi_Label *)`
- `Fl_Multi_Label::label(Fl_Menu_Item *)`
12.86.22 label() [2/3]

```cpp
void Fl_Menu_Item::label (const char * a) [inline]
```

Sets the title (label) of the menu item.

**Note**

This does **not** change the `labeltype()` for backwards compatibility. Take care to assign the correct `labeltype()` if you assign different label types to the same menu item sequentially.

The default `labeltype()` is `FL_NORMAL_LABEL`.

See also

```cpp
label(Fl_Labeltype, const char*)
```

12.86.23 label() [3/3]

```cpp
void Fl_Menu_Item::label (Fl_Labeltype a, const char * b) [inline]
```

Sets the title (label) and the label type of the menu item.

The default `Fl_Labeltype` when using `Fl_Menu_Item::label(const char * a)` is `FL_NORMAL_LABEL` but you can set any other label type, for instance `FL_EMBOSSED_LABEL`.

Special labeltypes are:

- `FL_ICON_LABEL`: draws the icon (`Fl_Image`) associated with the text
- `FL_IMAGE_LABEL`: draws the image (`Fl_Image`) associated with the text
- `FL_MULTI_LABEL`: draws multiple parts side by side, see `Fl_Multi_Label`

See also

```cpp
const char* Fl_Menu_Item::label() const
```

12.86.24 labelcolor() [1/2]

```cpp
Fl_Color Fl_Menu_Item::labelcolor () const [inline]
```

Gets the menu item's label color.

This color is passed to the labeltype routine, and is typically the color of the label text. This defaults to `FL_BLACK`. If this color is not black fltk will **not** use overlay bitplanes to draw the menu - this is so that images put in the menu draw correctly.

12.86.25 labelcolor() [2/2]

```cpp
void Fl_Menu_Item::labelcolor (Fl_Color a) [inline]
```

Sets the menu item's label color.

See also

```cpp
Fl_Color Fl_Menu_Item::labelcolor() const
```

12.86.26 labelfont() [1/2]

```cpp
Fl_Font Fl_Menu_Item::labelfont () const [inline]
```

Gets the menu item's label font.

Fonts are identified by small 8-bit indexes into a table. See the enumeration list for predefined fonts. The default value is a Helvetica font. The function `Fl::set_font()` can define new fonts.
12.86.27  labelfont() [2/2]

void Fl_Menu_Item::labelfont (  
    Fl_Font a ) [inline]

Sets the menu item's label font.
Fonts are identified by small 8-bit indexes into a table. See the enumeration list for predefined fonts. The default value is a Helvetica font. The function Fl::set_font() can define new fonts.

12.86.28  labeltype() [1/2]

Fl_Labeltype Fl_Menu_Item::labeltype ( ) const [inline]

Returns the menu item's labeltype.
A labeltype identifies a routine that draws the label of the widget. This can be used for special effects such as emboss, or to use the label() pointer as another form of data such as a bitmap. The value FL_NORMAL_LABEL prints the label as text.

12.86.29  labeltype() [2/2]

void Fl_Menu_Item::labeltype (  
    Fl_Labeltype a ) [inline]

Sets the menu item's labeltype.
A labeltype identifies a routine that draws the label of the widget. This can be used for special effects such as emboss, or to use the label() pointer as another form of data such as a bitmap. The value FL_NORMAL_LABEL prints the label as text.

12.86.30  measure()

int Fl_Menu_Item::measure (  
    int * hp,  
    const Fl_Menu_ * m ) const

Measures width of label, including effect of & characters. Optionally, can get height if hp is not NULL.

12.86.31  multi_label()

void Fl_Menu_Item::multi_label (  
    const Fl_Multi_Label * ml ) [inline]

Sets the title (label()) and labeltype() to an Fl_Multi_Label.
This sets the labeltype() to _FL_MULTI_LABEL (note the leading underscore).
See also  

const char* Fl_Menu_Item::label() const

Since

1.4.0

12.86.32  next() [1/2]

Fl_Menu_Item * Fl_Menu_Item::next (  
    int i = 1 ) [inline]

Advances a pointer by n items through a menu array, skipping the contents of submenus and invisible items. There are two calls so that you can advance through const and non-const data.

12.86.33  next() [2/2]

const Fl_Menu_Item * Fl_MENU::next (  
    int n = 1 ) const

Advance a pointer by n items through a menu array, skipping the contents of submenus and invisible items. There are two calls so that you can advance through const and non-const data.
12.86.2.34 popup()

const Fl_Menu_Item * Fl_Menu_Item::popup (  
    int X,  
    int Y,  
    const char * title = 0,  
    const Fl_Menu_Item * picked = 0,  
    const Fl_Menu_Item * menu_button = 0 ) const

This method is called by widgets that want to display menus. The menu stays up until the user picks an item or dismisses it. The selected item (or NULL if none) is returned. This does not do the callbacks or change the state of check or radio items. The menu is positioned so the cursor is centered over the item picked. This will work even if picked is in a submenu. If picked is zero or not in the menu item table the menu is positioned with the cursor in the top-left corner.

Parameters

| in | X,Y | the position of the mouse cursor, relative to the window that got the most recent event (usually you can pass Fl::event_x() and Fl::event_y() unchanged here). |
| in | title | a character string title for the menu. If non-zero a small box appears above the menu with the title in it. |
| in | picked | if this pointer is not NULL, the popup menu will appear so that the picked menu is under the mouse pointer. |
| in | menu_button | is a pointer to an Fl_Menu_ from which the color and boxtypes for the menu are pulled. If NULL then defaults are used. |

Returns

a pointer to the menu item selected by the user, or NULL

12.86.2.35 pulldown()

const Fl_Menu_Item * Fl_Menu_Item::pulldown (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const Fl_Menu_Item * initial_item = 0,  
    const Fl_Menu_Item * pbutton = 0,  
    const Fl_Menu_Item * title = 0,  
    int menubar = 0 ) const

Pulldown() is similar to popup(), but a rectangle is provided to position the menu. The menu is made at least W wide, and the picked item initial_item is centered over the rectangle (like Fl_Choice uses). If initial_item is NULL or not found, the menu is aligned just below the rectangle (like a pulldown menu). The title and menubar arguments are used internally by the Fl_Menu_Bar widget.

12.86.2.36 radio()

int Fl_Menu_Item::radio ( ) const [inline]

Returns true if this item is a radio item. When a radio button is selected all "adjacent" radio buttons are turned off. A set of radio items is delimited by an item that has radio() false, or by an item with FL_MENU_DIVIDER turned on.

12.86.2.37 set()

void Fl_Menu_Item::set ( ) [inline]

Turns the check or radio item "on" for the menu item. Note that this does not turn off any adjacent radio items like setonly() does.
12.86.2.38  setonly()

```cpp
void Fl_Menu_Item::setonly(
    Fl_Menu_Item const * first = NULL)
```

Turns the radio item "on" for the menu item and turns "off" adjacent radio items set.

**Note**
This method is dangerous if radio items are first in the menu. Make sure that `first` is set correctly or use `Fl_Menu_::setonly(Fl_Menu_Item*)` instead.

**Parameters**
- `in first` start of menu array or NULL (default) if the radio group is not the first item

12.86.2.39  shortcut()

```cpp
void Fl_Menu_Item::shortcut (int s) [inline]
```

Sets exactly what key combination will trigger the menu item. The value is a logical 'or' of a key and a set of shift flags, for instance FL_ALT+'a' or FL_ALT+FL_F+10 or just 'a'. A value of zero disables the shortcut. The key can be any value returned by `Fl::event_key()`, but will usually be an ASCII letter. Use a lower-case letter unless you require the shift key to be held down. The shift flags can be any set of values accepted by `Fl::event_state()`. If the bit is on that shift key must be pushed. Meta, Alt, Ctrl, and Shift must be off if they are not in the shift flags (zero for the other bits indicates a "don't care" setting).

12.86.2.40  size()

```cpp
int Fl_Menu_Item::size ( ) const
```

Size of the menu starting from this menu item. This method counts all menu items starting with this menu item, including all menu items in the same (sub)menu level, all nested submenus, and the terminating empty (0) menu item. It does not count menu items referred to by `FL_SUBMENU_POINTER` menu items (except the single menu item with `FL_SUBMENU_POINTER`). All menu items counted are consecutive in memory (one array).

**Example:**
```cpp
schemechoice = new Fl_Choice(X+125,Y,140,25,"FLTK Scheme");
schemechoice->add("none");
schemechoice->add("plastic");
schemechoice->add("gtk+");
printf("schemechoice->menu()->size() = %d\n", schemechoice->menu()->size());
```

Output:
```
schemechoice->menu()->size() = 5
```

12.86.2.41  submenu()

```cpp
int Fl_Menu_Item::submenu ( ) const [inline]
```

Returns true if either `FL_SUBMENU` or `FL_SUBMENU_POINTER` is on in the flags. `FL_SUBMENU` indicates an embedded submenu that goes from the next item through the next one with a NULL label(). `FL_SUBMENU_POINTER` indicates that `user_data()` is a pointer to another menu array.

12.86.2.42  test_shortcut()

```cpp
const Fl_Menu_Item * Fl_Menu_Item::test_shortcut ( ) const
```

This is designed to be called by a widgets handle() method in response to a FL_SHORTCUT event. If the current event matches one of the items shortcut, that item is returned. If the keystroke does not match any shortcuts then NULL is returned. This only matches the `shortcut()` fields, not the letters in the title preceeded by `'}'

---

Generated by Doxygen
12.86.2.43 uncheck()

void Fl_Menu_Item::uncheck () [inline]
Back compatibility only.

Deprecated Please use Fl_Menu_Item::clear() instead. This method will be removed in FLTK 1.5.0 or later.

See also
clear()

12.86.2.44 value()

int Fl_Menu_Item::value () const [inline]
Returns the current value of the check or radio item.
This is zero (0) if the menu item is not checked and non-zero otherwise.
Since

1.4.0 this method returns 1 if the item is checked but you should not rely on a particular value, only zero or non-zero.

Note
The returned value for a checked menu item was FL_MENU_VALUE (4) before FLTK 1.4.0.

The documentation for this struct was generated from the following files:
- Fl_Menu_Item.H
- Fl_Menu.cxx
- Fl_Menu-debug-disabled.cxx
- Fl_Menu-delete-labels.cxx
- Fl_Menu.cxx
- Fl_Menu_add.cxx

12.87 Fl_Menu_Window Class Reference

The Fl_Menu_Window widget is a window type used for menus.
#include <Fl_Menu_Window.H>
Inheritance diagram for Fl_Menu_Window:

public Member Functions

- Fl_Menu_Window (int W, int H, const char *l=0)
  Creates a new Fl_Menu_Window widget using the given size, and label string.
- Fl_Menu_Window (int X, int Y, int W, int H, const char *l=0)
  Creates a new Fl_Menu_Window widget using the given position, size, and label string.
- ~Fl_Menu_Window ()
  Destroys the window and all of its children.
Public Member Functions inherited from Fl_Single_Window

• **Fl_Single_Window** (int W, int H, const char ∗l=0)
  
  Creates a new Fl_Single_Window widget using the given size, and label (title) string.

• **Fl_Single_Window** (int X, int Y, int W, int H, const char ∗l=0)
  
  Creates a new Fl_Single_Window widget using the given position, size, and label (title) string.

• void show () FL_OVERRIDE
  
  Makes a widget visible.

• void show (int a, char ∗∗b)
  
  Same as Fl_Window::show(int a, char ∗∗b)

Public Member Functions inherited from Fl_Window

• virtual class Fl_Double_Window ∗ as_double_window ()
  
  Return non-null if this is an Fl_Double_Window object.

• virtual class Fl_Overlay_Window ∗ as_overlay_window ()
  
  Return non-null if this is an Fl_Overlay_Window object.

• Fl_Window const ∗ as_window () const FL_OVERRIDE

• Fl_Window ∗ as_window () FL_OVERRIDE
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• unsigned int border () const
  
  Returns whether the window possesses a border.

• void border (int b)
  
  Sets whether or not the window manager border is around the window.

• void clear_border ()
  
  Fast inline function to turn the window manager border off.

• void clear_modal_states ()
  
  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.

• void copy_label (const char ∗a)
  
  Sets the window titlebar label to a copy of a character string.

• void cursor (const Fl_RGB_Image ∗, int, int)
  
  Changes the cursor for this window using the provided image as cursor's shape.

• void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  
  For back compatibility only.

• void cursor (Fl_Cursor)
  
  Changes the cursor for this window.

• int decorated_h () const
  
  Returns the window height including any window title bar and any frame added by the window manager.

• int decorated_w () const
  
  Returns the window width including any frame added by the window manager.

• void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  
  For back compatibility only.

• void default_cursor (Fl_Cursor)
  
  Sets the default window cursor.

• void draw_backdrop ()
  
  Draw the background image if one is set and is aligned inside.

• Fl_Window (int w, int h, const char ∗title=0)
  
  Creates a window from the given width w, height h, and title.

• Fl_Window (int x, int y, int w, int h, const char ∗title=0)
  
  Creates a window from the given position (x, y), size (w, h) and title.

• void free_position ()
Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.

- **void fullscreen()**
  Makes the window completely fill one or more screens, without any window manager border visible.

- **unsigned int fullscreen_active() const**
  Returns non zero if FULLSCREEN flag is set, 0 otherwise.

- **void fullscreen_off()**
  Turns off any side effects of fullscreen()

- **void fullscreen_off(int X, int Y, int W, int H)**
  Turns off any side effects of fullscreen() and does resize(x,y,w,h).

- **void fullscreen_screens(int top, int bottom, int left, int right)**
  Sets which screens should be used when this window is in fullscreen mode.

- **int handle(int) FL_OVERRIDE**
  Handles the specified event.

- **void hide() FL_OVERRIDE**
  Removes the window from the screen.

- **void hotspot(const Fl_Widget &p, int offscreen=0)**
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

- **void hotspot(const Fl_Widget *, int offscreen=0)**
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

- **void hotspot(int x, int y, int offscreen=0)**
  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.

- **const void *icon() const**
  Gets the current icon window target dependent data.

- **void icon(const Fl_RGB_Image *)**
  Sets or resets a single window icon.

- **void icon(const void *ic)**
  Platform-specific method to set the window icon usable on Windows and X11 only.

- **void iconize()**
  Iconifies the window.

- **const char *iconlabel() const**
  See void Fl_Window::iconlabel(const char *)

- **void iconlabel(const char *)**
  Sets the icon label.

- **void icons(const Fl_RGB_Image *[])**, int
  Sets the window icons.

- **void icons(HICON big_icon, HICON small_icon)**
  Sets the window icons using HICON handles (Windows platform only).

- **const char *label() const**
  See void Fl_Window::label(const char *)

- **void label(const char *)**
  Sets the window title bar label.

- **void label(const char *label, const char *iconlabel)**
  Sets the icon label.

- **void make_current()**
  Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.

- **void maximize()**
  Maximizes a top-level window to its current screen.
• unsigned int maximize_active () const
  Returns whether the window is currently maximized.

• unsigned int menu_window () const
  Returns true if this window is a menu window.

• unsigned int modal () const
  Returns true if this window is modal.

• unsigned int non_modal () const
  Returns true if this window is modal or non-modal.

• fl_uintptr_t os_id ()
  Returns a platform-specific identification of a shown window, or 0 if not shown.

• unsigned int override () const
  Returns non zero if OVERRIDE flag is set, 0 otherwise.

• void resize (int X, int Y , int W, int H) FL_OVERRIDE
  Changes the size and position of the window.

• int screen_num ()
  The number of the screen containing the mapped window.

• void screen_num (int screen_num)
  Set the number of the screen where to map the window.

• void set_menu_window ()
  Marks the window as a menu window.

• void set_modal ()
  A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program,
  and will also remain on top of the other windows (if the X window manager supports the "transient for" property).

• void set_non_modal ()
  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on
  top, but it has no effect on event delivery.

• void set_override ()
  Activates the flags NOBORDER|OVERRIDE.

• void set_tooltip_window ()
  Marks the window as a tooltip window.

• const Fl_Ima χ∗ shape ()
  Returns the image controlling the window shape or NULL.

• void shape (const Fl_Ima χ &b)
  Set the window's shape with an Fl_Ima χ.

• void shape (const Fl_Ima χ *img)
  Assigns a non-rectangular shape to the window.

• void show () FL_OVERRIDE
  Puts the window on the screen.

• void show (int argc, char ∗ ∗argv)
  Puts the window on the screen with show() and parses command-line arguments.

• int shown ()
  Returns non-zero if show() has been called (but not hide()).

• void size_range (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)
  Sets the allowable range the user can resize this window to.

• unsigned int tooltip_window () const
  Returns true if this window is a tooltip window.

• void un_maximize ()
  Returns a previously maximized top-level window to its previous size.

• void wait_for_expose ()
  Waits for the window to be displayed after calling show().

• int x_root () const
Gets the x position of the window on the screen.

- const char * xclass () const
  Returns the xclass for this window, or a default.

- void xclass (const char *c)
  Sets the xclass for this window.

- int y_root () const
  Gets the y position of the window on the screen.

- virtual ~FL_Window ()
  The destructor also deletes all the children.

Public Member Functions inherited from FL_Group

- FL_Widget *& _ddfdesign_kludge ()
  This is for forms compatibility only.

- void add (FL_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.

- void add (FL_Widget *o)
  See void FL_Group::add(FL_Widget &w)

- void addResizable (FL_Widget &o)
  Adds a widget to the group and makes it the resizable widget.

- FL_Widget *const * array () const
  Returns a pointer to the array of children.

- Fl_Group const * as_group () const FL_OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- void begin ()
  Sets the current group so you can build the widget tree by just constructing the widgets.

- FL_Widget * child (int n) const
  Returns array()[n].

- int children () const
  Returns how many child widgets the group has.

- void clear ()
  Deletes all child widgets from memory recursively.

- unsigned int clip_children ()
  Returns the current clipping mode.

- void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.

- void end ()
  Exactly the same as current(this->parent()).

- int find (const FL_Widget &o) const
  See int FL_Group::find(const FL_Widget *w) const.

- int find (const FL_Widget *) const
  Searches the child array for the widget and returns the index.

- FL_Group (int, int, int, int, const char * = 0)
  Creates a new FL_Group widget using the given position, size, and label string.

- void focus (FL_Widget *W)

- void forms_end ()
  This is for forms compatibility only.

- void init_sizes ()

Generated by Doxygen
Resets the internal array of widget sizes and positions.

- **void** `insert(Fl_Widget &, int i)`
  The widget is removed from its current group (if any) and then inserted into this group.

- **void** `insert(Fl_Widget &, Fl_Widget *)before`
  This does `insert(w, find(before))`.

- **void** `remove(Fl_Widget &)`
  Removes a widget from the group but does not delete it.

- **void** `remove(Fl_Widget *o)`
  Removes the widget `o` from the group.

- **void** `remove(int index)`
  Removes the widget at `index` from the group but does not delete it.

- **Fl_Widget **`resizable()` const
  Returns the group’s resizable widget.

- **void** `resizable(Fl_Widget &o)`
  Sets the group’s resizable widget.

- **void** `resizable(Fl_Widget *o)`
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- **virtual** `~Fl_Group ()`
  The destructor also deletes all the children.

**Public Member Functions inherited from Fl_Widget**

- **void** `_clear_fullscreen ()`
- **void** `_set_fullscreen ()`
- **void** `activate ()`
  Activates the widget.

- **unsigned int** `active () const`
  Returns whether the widget is active.

- **int** `active_r () const`
  Returns whether the widget and all of its parents are active.

- **Fl_Align** `align () const`
  Gets the label alignment.

- **void** `align(Fl_Align alignment)`
  Sets the label alignment.

- **long** `argument () const`
  Gets the current user data (long) argument that is passed to the callback function.

- **void** `argument (long v)`
  Sets the current user data (long) argument that is passed to the callback function.

- **virtual class Fl_Gl_Window **`as_gl_window ()`
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- **virtual class Fl_Gl_Window const **`as_gl_window () const`

- **void** `bind_deimage (Fl_Image *img)`
  Sets the image to use as part of the widget label when in the inactive state.

- **void** `bind_deimage (int f)`
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- **void** `bind_image (Fl_Image *img)`
  Sets the image to use as part of the widget label when in the active state.

- **void** `bind_image (int f)`
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype** `box () const`
  Gets the box type of the widget.
• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.

• Fl_Callback_p callback () const
  Gets the current callback function for the widget.

• void callback (Fl_Callback +cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback +cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

• void callback (Fl_Callback +cb, void *p)
  Sets the current callback function and data for the widget.

• void callback (Fl_Callback0 +cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback1 +cb, long p=0)
  Sets the current callback function for the widget.

• unsigned int changed () const
  Checks if the widget value changed since the last callback.

• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

• void clear_changed ()
  Marks the value of the widget as unchanged.

• void clear_damage (uchar c=0)
  Clears or sets the damage flags.

• void clear_output ()
  Sets a widget to accept input.

• void clear_visible ()
  Hides the widget.

• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

• Fl_Color color () const
  Gets the background color of the widget.

• void color (Fl_Color bg)
  Sets the background color of the widget.

• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

• Fl_Color color2 () const
  For back compatibility only.

• void color2 (unsigned a)
  For back compatibility only.

• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.

• void copy_label (const char *new_label)
  Sets the current label.

• void copy_tooltip (const char *text)
  Sets the current tooltip text.

• uchar damage () const
  Returns non-zero if draw() needs to be called.

• void damage (uchar c)
  Sets the damage bits for the widget.

• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

• int damage_resize (int, int, int, int)
Internal use only.

- **void deactivate ()**
  Deactivates the widget.

- **Fl_Image * deimage ()**
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image * deimage () const**
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (Fl_Image &img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const**
  Returns whether the inactive image is managed by the widget.

- **void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with default arguments.

- **void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void draw_label (int, int, int, int, Fl_Align) const**
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h () const**
  Gets the widget height.

- **Fl_Image * image ()**
  Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image * image () const**
  Gets the image that is used as part of the widget label when in the active state.

- **void image (Fl_Image &img)**
  Sets the image to use as part of the widget label when in the active state.

- **void image (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const**
  Returns whether the image is managed by the widget.

- **int inside (const Fl_Widget *wgt) const**
  Checks if this widget is a child of wgt.

- **int is_label_copied () const**
  Returns whether the current label was assigned with copy_label().

- **const char * label () const**
  Gets the current label text.

- **void label (const char *text)**
  Sets the current label pointer.

- **void label (Fl_Labeltype a, const char *b)**
  Shortcut to set the label text and type in one call.

- **FL_Color labelcolor () const**
  Gets the label color.

- **void labelcolor (FL_Color c)**
  Sets the label color.

- **FL_Font labelfont () const**
  Gets the font to use.

- **void labelfont (FL_Font f)**
  Sets the font to use.
• `Fl_Fontsize labelsize () const`
  Gets the font size in pixels.

• `void labelsize (Fl_Fontsize pix)`
  Sets the font size in pixels.

• `Fl_Labeltype labeltype () const`
  Gets the label type.

• `void labeltype (Fl_Labeltype a)`
  Sets the label type.

• `void measure_label (int &ww, int &hh) const`
  Sets width ww and height hh accordingly with the label size.

• `bool needs_keyboard () const`
  Returns whether this widget needs a keyboard.

• `void needs_keyboard (bool needs)`
  Sets whether this widget needs a keyboard.

• `unsigned int output () const`
  Returns if a widget is used for output only.

• `Fl_Group * parent () const`
  Returns a pointer to the parent widget.

• `void parent (Fl_Group * p)`
  Internal use only - “for hacks only”.

• `void position (int X, int Y)`
  Repositions the window or widget.

• `void redraw ()`
  Schedules the drawing of the widget.

• `void redraw_label ()`
  Schedules the drawing of the label.

• `Fl_Color selection_color () const`
  Gets the selection color.

• `void selection_color (Fl_Color a)`
  Sets the selection color.

• `void set_active ()`
  Marks the widget as active without sending events or changing focus.

• `void set_changed ()`
  Marks the value of the widget as changed.

• `void set_output ()`
  Sets a widget to output only.

• `void set_visible ()`
  Makes the widget visible.

• `void set_visible_focus ()`
  Enables keyboard focus navigation with this widget.

• `int shortcut_label () const`
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.

• `void shortcut_label (int value)`
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.

• `void size (int W, int H)`
  Changes the size of the widget.

• `int take_focus ()`
  Gives the widget the keyboard focus.

• `unsigned int takesevents () const`
  Returns if the widget is able to take events.

• `int testShortcut ()`
Returns true if the widget's label contains the entered '\&x' shortcut.

- **const char * tooltip () const**
  
  Gets the current tooltip text.

- **void tooltip (const char *text)**
  
  Sets the current tooltip text.

- **FL_Window * top_window () const**
  
  Returns a pointer to the top-level window for the widget.

- **FL_Window * top_window_offset (int &xoff, int &yoff) const**
  
  Finds the x/y offset of the current widget relative to the top-level window.

- **uchar type () const**
  
  Gets the widget type.

- **void type (uchar t)**
  
  Sets the widget type.

- **int use_accents_menu ()**
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- **void * user_data () const**
  
  Gets the user data for this widget.

- **void user_data (FL_Callback_User_Data *v, bool auto_free)**
  
  Sets the user data for this widget.

- **void user_data (void *v)**
  
  Sets the user data for this widget.

- **unsigned int visible () const**
  
  Returns whether a widget is visible.

- **unsigned int visible_focus () const**
  
  Checks whether this widget has a visible focus.

- **void visible_focus (int v)**
  
  Modifies keyboard focus navigation.

- **int visible_r () const**
  
  Returns whether a widget and all its parents are visible.

- **int w () const**
  
  Gets the widget width.

- **FL_When when () const**
  
  Returns the conditions under which the callback is called.

- **void when (uchar i)**
  
  Sets the flags used to decide when a callback is called.

- **FL_Window * window () const**
  
  Returns a pointer to the nearest parent window up the widget hierarchy.

- **int x () const**
  
  Gets the widget position in its window.

- **int y () const**
  
  Gets the widget position in its window.

- **virtual ~FL_Widget ()**
  
  Destroys the widget.

**Additional Inherited Members**

**Public Types inherited from FL_Window**

- **typedef struct HICON__ * HICON**
Static Public Member Functions inherited from Fl_Window

- static Fl_Window * current ()
  Returns the last window that was made current.
- static void default_callback (Fl_Window *, void *)
  Back compatibility: Sets the default callback v for win to call on close event.
- static void default_icon (const Fl_RGB_Image *)
  Sets a single default window icon.
- static void default_icons (const Fl_RGB_Image [ ], int)
  Sets the default window icons.
- static void default_icons (HICON big_icon, HICON small_icon)
  Sets the default window icons (Windows platform only).
- static const char * default_xclass ()
  Returns the default xclass.
- static void default_xclass (const char *)
  Sets the default window xclass.
- static bool is_a_rescale ()
  Returns true when a window is being rescaled.
- static char show_next_window_iconic ()
  Returns the static flag whether the next window should be opened iconified.
- static void show_next_window_iconic (char stat)
  Sets a static flag whether the next window should be opened iconified.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()
  Returns the currently active group.
- static void current (Fl_Group *g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *, void *)
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered 'x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , Copied_Tooltip = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.
Protected Member Functions inherited from Fl_Window

- void default_size_range ()
  Protected method to calculate the default size range of a window.
- void draw () FL_OVERRIDE
  Draws the widget.
- virtual void flush ()
  Forces the window to be drawn, this window is also made current and calls draw().
- int force_position () const
  Returns the internal state of the window's FORCE_POSITION flag.
- void force_position (int force)
  Sets an internal flag that tells FLTK and the window manager to honor position requests.
- void free_icons ()
  Deletes all icons previously attached to the window.
- int is_resizable ()
  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.
- void draw_children ()
  Draws all children of the group.
- void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)
  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()
  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  Draws a focus rectangle around the widget.
12.88 Fl_Multi_Browser Class Reference

- void **draw_focus** (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
- void **draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void **draw_label** () const
  Draws the widget's label at the defined label position.
- void **draw_label** (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.
- unsigned int **flags** () const
  Gets the widget flags mask.
- void h (int v)
  Internal use only.
- void **set_flag** (unsigned int c)
  Sets a flag in the flags mask.
- void w (int v)
  Internal use only.
- void x (int v)
  Internal use only.
- void y (int v)
  Internal use only.

**Static Protected Attributes inherited from Fl_Window**

- static Fl_Widget ∗current_
  Stores the last window that was made current.

12.87.1 Detailed Description

The **Fl_Menu_Window** widget is a window type used for menus. By default the window is drawn in the hardware overlay planes if they are available so that the menu don't force the rest of the window to redraw.

The documentation for this class was generated from the following files:
- Fl_Menu_Window.H
- Fl_Menu_Window.cxx

12.88 Fl_Multi_Browser Class Reference

The **Fl_Multi_Browser** class is a subclass of Fl_Browser which lets the user select any set of the lines.

#include <Fl_Multi_Browser.H>

Inheritance diagram for Fl_Multi_Browser:

```
Fl_Widget
   |
   v
Fl_Group
   |
   v
Fl_Browser_
   |
   v
Fl_Browser
   |
   v
Fl_Multi_Browser
```
Public Member Functions

• **Fl_Multi_Browser** (int X, int Y, int W, int H, const char ∗L=0)
  
  Creates a new Fl_Multi_Browser widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Browser

• void **add** (const char ∗newtext, void ∗d=0)
  
  Adds a new line to the end of the browser.

• void **bottomline** (int line)
  
  Scrolls the browser so the bottom item in the browser is showing the specified line.

• void **clear** ()
  
  Removes all the lines in the browser.

• char **column_char** () const
  
  Gets the current column separator character.

• void **column_char** (char c)
  
  Sets the column separator to c.

• const int ∗**column_widths** () const
  
  Gets the current column width array.

• void **column_widths** (const int ∗arr)
  
  Sets the current array to arr.

• void ∗**data** (int line) const
  
  Returns the user data() for specified line.

• void **data** (int line, void ∗d)
  
  Sets the user data for specified line to d.

• void **display** (int line, int val=1)
  
  For back compatibility.

• int **displayed** (int line) const
  
  Returns non-zero if line has been scrolled to a position where it is being displayed.

• **Fl_Browser** (int X, int Y, int W, int H, const char ∗L=0)
  
  The constructor makes an empty browser.

• char **format_char** () const
  
  Gets the current format code prefix character, which by default is '@'.

• void **format_char** (char c)
  
  Sets the current format code prefix character to c.

• void **hide** () FL_OVERRIDE
  
  Hides the entire Fl_Browser widget – opposite of show().

• void **hide** (int line)
  
  Make the item at the specified line invisible, preventing selection by the user.

• **Fl_Image** ∗**icon** (int line) const
  
  Returns the icon currently defined for line.

• void **icon** (int line, Fl_Image ∗icon)
  
  Set the image icon for line to the value icon.

• void **insert** (int line, const char ∗newtext, void ∗d=0)
  
  Insert a new entry whose label is newtext above given line, optional data d.

• void **lineposition** (int line, Fl_Line_Position pos)
  
  Updates the browser so that line is shown at position pos.

• int **load** (const char ∗filename)
  
  Clears the browser and reads the file, adding each line from the file to the browser.

• void **make_visible** (int line)
  
  Make the item at the specified line visible().

• void **middleline** (int line)
Scrolls the browser so the middle item in the browser is showing the specified line.

- **void move**(int to, int from)
  
  Line from is removed and reinserted at to.

- **void remove**(int line)
  
  Remove entry for given line number, making the browser one line shorter.

- **void remove_icon**(int line)
  
  Removes the icon for line.

- **void replace**(int a, const char ∗b)
  
  For back compatibility only.

- **int select**(int line, int val=1)
  
  Sets the selection state of the item at line to the value val.

- **int selected**(int line) const
  
  Returns 1 if specified line is selected, 0 if not.

- **void show ()** FL_OVERRIDE
  
  Shows the entire Fl_Browser widget – opposite of hide().

- **void show**(int line)
  
  Makes line visible, and available for selection by user.

- **int size ()** const
  
  Returns how many lines are in the browser.

- **void size**(int W, int H)

- **void swap**(int a, int b)
  
  Swaps two browser lines a and b.

- **const char ∗text**(int line) const
  
  Returns the label text for the specified line.

- **void text**(int line, const char ∗newtext)
  
  Sets the text for the specified line to newtext.

- **Fl_Fontsize textsize ()** const
  
  Gets the default text size (in pixels) for the lines in the browser.

- **void textsize**(Fl_Fontsize newSize)
  
  Sets the default text size (in pixels) for the lines in the browser to newSize.

- **int topline ()** const
  
  Returns the line that is currently visible at the top of the browser.

- **void topline**(int line)
  
  Scrolls the browser so the top item in the browser is showing the specified line.

- **int value ()** const
  
  Returns the line number of the currently selected line, or 0 if none selected.

- **void value**(int line)
  
  Sets the browser’s value(), which selects the specified line.

- **int visible**(int line) const
  
  Returns non-zero if the specified line is visible, 0 if hidden.

- **~Fl_Browser ()**
  
  The destructor deletes all list items and destroys the browser.

Public Member Functions inherited from Fl_Browser_

- **int deselect**(int docallbacks=0)
  
  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.

- **void display**(void ∗item)
  
  Displays the item, scrolling the list as necessary.

- **int handle**(int event) FL_OVERRIDE
  
  Handles the event within the normal widget bounding box.
• uchar **has_scrollbar () const**
  
  Returns the current scrollbar mode, see Fl_Browser_::has_scrollbar(uchar)

• void **has_scrollbar (uchar mode)**
  
  Sets whether the widget should have scrollbars or not (default Fl_Browser_::BOTH).

• int **hposition () const**
  
  Gets the horizontal scroll position of the list as a pixel position pos.

• void **hposition (int)**
  
  Sets the horizontal scroll position of the list to pixel position pos.

• int **linespacing () const**
  
  Return the height of additional spacing between browser lines.

• void **linespacing (int pixels)**
  
  Add some space between browser lines.

• int **position () const**
  
  • void position (int pos)
  
  • void position (int x, int y)

• void **resize (int X, int Y, int W, int H) FL_OVERRIDE**
  
  Repositions and/or resizes the browser.

• void **scrollbar_left ()**
  
  Moves the vertical scrollbar to the lefthand side of the list.

• void **scrollbar_right ()**
  
  Moves the vertical scrollbar to the righthand side of the list.

• int **scrollbar_size () const**
  
  Gets the current size of the scrollbars' troughs, in pixels.

• void **scrollbar_size (int newSize)**
  
  Sets the pixel size of the scrollbars' troughs to newSize, in pixels.

• int **scrollbar_width () const**
  
  Returns the global value Fl::scrollbar_size().

• void **scrollbar_width (int width)**
  
  Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

• int **select (void ∗item, int val=1, int docallbacks=0)**
  
  Sets the selection state of item to val, and returns 1 if the state changed or 0 if it did not.

• int **select_only (void ∗item, int docallbacks=0)**
  
  Selects item and returns 1 if the state changed or 0 if it did not.

• void **sort (int flags=0)**
  
  Sort the items in the browser based on flags.

• Fl_Color **textcolor () const**
  
  Gets the default text color for the lines in the browser.

• void **textcolor (Fl_Color col)**
  
  Sets the default text color for the lines in the browser to color col.

• Fl_Font **textfont () const**
  
  Gets the default text font for the lines in the browser.

• void **textfont (Fl_Font font)**
  
  Sets the default text font for the lines in the browser to font.

• Fl_Fontsize **textsize () const**
  
  Gets the default text size (in pixels) for the lines in the browser.

• void **textsize (Fl_Fontsize newSize)**
  
  Sets the default text size (in pixels) for the lines in the browser to size.

• int **vposition () const**
  
  Gets the vertical scroll position of the list as a pixel position pos.

• void **vposition (int pos)**
  
  Sets the vertical scroll position of the list to pixel position pos.
Public Member Functions inherited from Fl_Group

- `Fl_Widget * & ddfdesign_kludge ()`
  
  This is for forms compatibility only.

- `void add (Fl_Widget &)`

  The widget is removed from its current group (if any) and then added to the end of this group.

- `void add (Fl_Widget *o)`

  See void Fl_Group::add(Fl_Widget &w)

- `void add_resizable (Fl_Widget &o)`

  Adds a widget to the group and makes it the resizable widget.

- `Fl_Widget * const * array () const`

  Returns a pointer to the array of children.

- `Fl_Group const * as_group () const FL_OVERRIDE`

  Returns an Fl_Group pointer if this widget is an Fl_Group.

- `void begin ()`

  Sets the current group so you can build the widget tree by just constructing the widgets.

- `Fl_Widget * child (int n) const`

  Returns array()[n].

- `int children () const`

  Returns how many child widgets the group has.

- `void clear ()`

  Deletes all child widgets from memory recursively.

- `unsigned int clip_children ()`

  Returns the current clipping mode.

- `void clip_children (int c)`

  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- `virtual int delete_child (int n)`

  Removes the widget at index from the group and deletes it.

- `void end ()`

  Exactly the same as current(this->parent()).

- `int find (const Fl_Widget &o) const`

  See int Fl_Group::find(const Fl_Widget *w) const.

- `int find (const Fl_Widget *) const`

  Searches the child array for the widget and returns the index.

- `Fl_Group (int, int, int, int, const char * = 0)`

  Creates a new Fl_Group widget using the given position, size, and label string.

- `void focus (Fl_Widget *)`

- `void forms_end ()`

  This is for forms compatibility only.

- `int handle (int) FL_OVERRIDE`

  Handles the specified event.

- `void init_sizes ()`

  Resets the internal array of widget sizes and positions.

- `void insert (Fl_Widget &, int i)`

  The widget is removed from its current group (if any) and then inserted into this group.

- `void insert (Fl_Widget &o, Fl_Widget *before)`

  This does insert(w, find(before)).

- `void remove (Fl_Widget &)`

  Removes a widget from the group but does not delete it.

- `void remove (Fl_Widget *o)`
Removes the widget \( o \) from the group.

- **void** remove (int index)
  
  Removes the widget at index from the group but does not delete it.

- **Fl_Widget** ∗ resizable () const
  
  Returns the group's resizable widget.

- **void** resizable (Fl_Widget &o)
  
  Sets the group's resizable widget.

- **void** resizable (Fl_Widget ∗o)
  
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- **void** resize (int, int, int, int) FL_OVERRIDE
  
  Resizes the Fl_Group widget and all of its children.

- **virtual ~Fl_Group ()**
  
  The destructor also deletes all the children.

### Public Member Functions inherited from Fl_Widget

- **void** _clear_fullscreen ()
- **void** _set_fullscreen ()
- **void** activate ()
  
  Activates the widget.

- **unsigned int** active () const
  
  Returns whether the widget is active.

- **int** active_r () const
  
  Returns whether the widget and all of its parents are active.

- **Fl_Align** align () const
  
  Gets the label alignment.

- **void** align (Fl_Align alignment)
  
  Sets the label alignment.

- **long** argument () const
  
  Gets the current user data (long) argument that is passed to the callback function.

- **void** argument (long v)
  
  Sets the current user data (long) argument that is passed to the callback function.

- **virtual class Fl_Gl_Window ∗ as_gl_window ()**
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- **virtual class Fl_Gl_Window const ∗ as_gl_window () const**

- **virtual Fl_Window ∗ as_window ()**
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- **virtual Fl_Window const ∗ as_window () const**

- **void** bind_deimage (Fl_Image ∗img)
  
  Sets the image to use as part of the widget label when in the inactive state.

- **void** bind_deimage (int f)
  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- **void** bind_image (Fl_Image ∗img)
  
  Sets the image to use as part of the widget label when in the active state.

- **void** bind_image (int f)
  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype** box () const
  
  Gets the box type of the widget.

- **void** box (Fl_Boxtype new_box)
  
  Sets the box type for the widget.

- **Fl_Callback_p** callback () const
Gets the current callback function for the widget.

- **void callback (Fl_Callback cb)**
  Sets the current callback function for the widget.

- **void callback (Fl_Callback cb, Fl_Callback_User_Data *p, bool auto_free)**
  Sets the current callback function and managed user data for the widget.

- **void callback (Fl_Callback cb, void *p)**
  Sets the current callback function and data for the widget.

- **void callback (Fl_Callback0 cb)**
  Sets the current callback function for the widget.

- **void callback (Fl_Callback1 cb, long p=0)**
  Sets the current callback function for the widget.

- **unsigned int changed () const**
  Checks if the widget value changed since the last callback.

- **void clear_active ()**
  Marks the widget as inactive without sending events or changing focus.

- **void clear_changed ()**
  Marks the value of the widget as unchanged.

- **void clear_damage (uchar c=0)**
  Clears or sets the damage flags.

- **void clear_output ()**
  Sets a widget to accept input.

- **void clear_visible ()**
  Hides the widget.

- **void clear_visible_focus ()**
  Disables keyboard focus navigation with this widget.

- **Fl_Color color () const**
  Gets the background color of the widget.

- **void color (Fl_Color bg)**
  Sets the background color of the widget.

- **void color (Fl_Color bg, Fl_Color sel)**
  Sets the background and selection color of the widget.

- **Fl_Color color2 () const**
  For back compatibility only.

- **void color2 (unsigned a)**
  For back compatibility only.

- **int contains (const Fl_Widget *w) const**
  Checks if w is a child of this widget.

- **void copy_label (const char *new_label)**
  Sets the current label.

- **void copy_tooltip (const char *text)**
  Sets the current tooltip text.

- **uchar damage () const**
  Returns non-zero if draw() needs to be called.

- **void damage (uchar c)**
  Sets the damage bits for the widget.

- **void damage (uchar c, int x, int y, int w, int h)**
  Sets the damage bits for an area inside the widget.

- **int damage_resize (int, int, int, int)**
  Internal use only.

- **void deactivate ()**
  Deactivates the widget.
• FL_Image * deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const FL_Image * deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (FL_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (FL_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• FL_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.
• const FL_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (FL_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (FL_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• FL_Color labelcolor () const
  Gets the label color.
• void labelcolor (FL_Color c)
  Sets the label color.
• FL_Font labelfont () const
  Gets the font to use.
• void labelfont (FL_Font f)
  Sets the font to use.
• FL_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (FL_Fontsize pix)
Sets the font size in pixels.

- `Fl_Labeltype labeltype () const`
  Gets the label type.

- `void labeltype (Fl_Labeltype a)`
  Sets the label type.

- `void measure_label (int &ww, int &hh) const`
  Sets width `ww` and height `hh` accordingly with the label size.

- `bool needs_keyboard () const`
  Returns whether this widget needs a keyboard.

- `void needs_keyboard (bool needs)`
  Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  Returns if a widget is used for output only.

- `Fl_Group * parent () const`
  Returns a pointer to the parent widget.

- `void parent (Fl_Group *p)`
  Internal use only - "for hacks only".

- `void position (int X, int Y)`
  Repositions the window or widget.

- `void redraw ()`
  Schedules the drawing of the widget.

- `void redraw_label ()`
  Schedules the drawing of the label.

- `Fl_Color selection_color () const`
  Gets the selection color.

- `void selection_color (Fl_Color a)`
  Sets the selection color.

- `void set_active ()`
  Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  Marks the value of the widget as changed.

- `void set_output ()`
  Sets a widget to output only.

- `void set_visible ()`
  Makes the widget visible.

- `void set_visible_focus ()`
  Enables keyboard focus navigation with this widget.

- `int shortcut_label () const`
  Returns whether the widget's label uses '&' to indicate shortcuts.

- `void shortcut_label (int value)`
  Sets whether the widget's label uses '&' to indicate shortcuts.

- `void size (int W, int H)`
  Changes the size of the widget.

- `int take_focus ()`
  Gives the widget the keyboard focus.

- `unsigned int takesevents () const`
  Returns if the widget is able to take events.

- `int testShortcut ()`
  Returns true if the widget's label contains the entered '&' shortcut.

- `const char * tooltip () const`
  Gets the current tooltip text.
• **void tooltip (const char ∗text)**
  
  Sets the current tooltip text.

• **Fl_Window ∗ top_window () const**
  
  Returns a pointer to the top-level window for the widget.

• **Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const**
  
  Finds the x/y offset of the current widget relative to the top-level window.

• **uchar type () const**
  
  Gets the widget type.

• **void type (uchar t)**
  
  Sets the widget type.

• **int use_accents_menu ()**
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• **void ∗ user_data () const**
  
  Gets the user data for this widget.

• **void user_data (Fl_Callback_User_Data ∗v, bool auto_free)**
  
  Sets the user data for this widget.

• **void user_data (void ∗v)**
  
  Sets the user data for this widget.

• **unsigned int visible () const**
  
  Returns whether a widget is visible.

• **unsigned int visible_focus () const**
  
  Checks whether this widget has a visible focus.

• **void visible_focus (int v)**
  
  Modifies keyboard focus navigation.

• **int visible_r () const**
  
  Returns whether a widget and all its parents are visible.

• **int w () const**
  
  Gets the widget width.

• **Fl_When when () const**
  
  Returns the conditions under which the callback is called.

• **void when (uchar i)**
  
  Sets the flags used to decide when a callback is called.

• **Fl_Window ∗ window () const**
  
  Returns a pointer to the nearest parent window up the widget hierarchy.

• **int x () const**
  
  Gets the widget position in its window.

• **int y () const**
  
  Gets the widget position in its window.

• **virtual ∼Fl_Widget ()**
  
  Destroys the widget.

---

**Additional Inherited Members**

**Public Types inherited from Fl_Browser**

• enum **Fl_Line_Position { TOP, BOTTOM, MIDDLE }**
  
  For internal use only?

**Public Types inherited from Fl_Browser_**

• enum {

  HORIZONTAL = 1, VERTICAL = 2, BOTH = 3, ALWAYS_ON = 4,
  HORIZONTAL_ALWAYS = 5, VERTICAL_ALWAYS = 6, BOTH_ALWAYS = 7
}

  Values for has_scrollbar().

---

Generated by Doxygen
Static Public Member Functions inherited from Fl_Group

• static Fl_Group * current ()
  Returns the currently active group.
• static void current (Fl_Group *g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
• static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
• static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Public Attributes inherited from Fl_Browser_

• Fl_Scrollbar hscrollbar
  Horizontal scrollbar.
• Fl_Scrollbar scrollbar
  Vertical scrollbar.

Protected Types inherited from Fl_Widget

• enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9,copied_label = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19, NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31,
} flags possible values enumeration.

Protected Member Functions inherited from Fl_Browser

• FL_BLINE * _remove (int line)
  Removes the item at the specified line.
• FL_BLINE * find_line (int line) const
  Returns the item for specified line.
• int full_height () const FL_OVERRIDE
  The height of the entire list of all visible() items in pixels.
• int incr_height () const FL_OVERRIDE
  The default 'average' item height (including inter-item spacing) in pixels.
• void insert (int line, FL_BLINE *item)
• void *item_at (int line) const FL_OVERRIDE
  Return the item at specified line.
• void item_draw (void *item, int X, int Y, int W, int H) const FL_OVERRIDE
Draws item at the position specified by X Y W H.

- **void** item_first () const FL_OVERRIDE
  Returns the very first item in the list.

- **int** item_height (void *item) const FL_OVERRIDE
  Returns height of item in pixels.

- **void** item_last () const FL_OVERRIDE
  Returns the very last item in the list.

- **void** item_next (void *item) const FL_OVERRIDE
  Returns the next item after item.

- **void** item_prev (void *item) const FL_OVERRIDE
  Returns the previous item before item.

- **void** item_select (void *item, int val) FL_OVERRIDE
  Change the selection state of item to the value val.

- **int** item_selected (void *item) const FL_OVERRIDE
  See if item is selected.

- **void** item_swap (void *a, void *b) FL_OVERRIDE
  Swap the items a and b.

- **const char** *item_text (void *item) const FL_OVERRIDE
  Returns the label text for item.

- **int** item_width (void *item) const FL_OVERRIDE
  Returns width of item in pixels.

- **int** lineno (void *item) const
  Returns line number corresponding to item, or zero if not found.

- **void** swap (FL_BLINE *a, FL_BLINE *b)
  Swap the two items a and b.

---

**Protected Member Functions inherited from Fl_Browser_**

- **void** bbox (int &X, int &Y, int &W, int &H) const
  Returns the bounding box for the interior of the list's display window, inside the scrollbars.

- **void** deleting (void *item)
  This method should be used when item is being deleted from the list.

- **int** displayed (void *item) const
  Returns non-zero if item has been scrolled to a position where it is being displayed.

- **void** draw () FL_OVERRIDE
  Draws the list within the normal widget bounding box.

- **void** *find_item (int ypos)
  Returns the item under mouse y position ypos.

- **Fl_Browser_** (int X, int Y, int W, int H, const char *L=0)
  The constructor makes an empty browser.

- **virtual int** full_width () const
  This method may be provided by the subclass to indicate the full width of the item list, in pixels.

- **void** inserting (void *a, void *b)
  This method should be used when an item is in the process of being inserted into the list.

- **virtual int** item_quick_height (void *item) const
  This method may be provided by the subclass to return the height of the item, in pixels.

- **int** leftedge () const
  This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.

- **void** new_list ()
  This method should be called when the list data is completely replaced or cleared.

- **void** redraw_line (void *item)
This method should be called when the contents of item has changed, but not its height.

- void redraw_lines ()
  This method will cause the entire list to be redrawn.
- void replacing (void ∗a, void ∗b)
  This method should be used when item a is being replaced by item b.
- void ∗selection () const
  Returns the item currently selected, or NULL if there is no selection.
- void swapping (void ∗a, void ∗b)
  This method should be used when two items a and b are being swapped.
- void ∗top () const
  Returns the item that appears at the top of the list.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
- void draw () FL_OVERRIDE
  Draws the widget.
- void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.
- void draw_children ()
  Draws all children of the group.
- void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget ∗, int)
  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)
  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.
- int ∗sizes ()
  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget’s position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h) const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
Draws a focus box for the widget at the given position and size.

- void draw_label() const
  Draws the widget's label at the defined label position.
- void draw_label(int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget(int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.

- unsigned int flags() const
  Gets the widget flags mask.
- void h(int v)
  Internal use only.
- void set_flag(unsigned int c)
  Sets a flag in the flags mask.
- void w(int v)
  Internal use only.
- void x(int v)
  Internal use only.
- void y(int v)
  Internal use only.

### 12.88.1 Detailed Description

The Fl_Multi_Browser class is a subclass of Fl_Browser which lets the user select any set of the lines.

![Fl_Multi_Browser](image)

Figure 12.31 Fl_Multi_Browser

The user interface is Macintosh style: clicking an item turns off all the others and selects that one, dragging selects all the items the mouse moves over, and ctrl + click (Cmd+click on the Mac OS platform) toggles the items. Shift + click extends the selection until the clicked item. This is different from how forms did it. Normally the callback is done when the user releases the mouse, but you can change this with when(). See Fl_Browser for methods to add and remove lines from the browser.

### 12.88.2 Constructor & Destructor Documentation

#### 12.88.2.1 Fl_Multi_Browser()

Fl_Multi_Browser::Fl_Multi_Browser(int X, int Y, int W, int H, const char *L = 0L)

Creates a new Fl_Multi_Browser widget using the given position, size, and label string.

The default boxtype is FL_DOWN_BOX. The constructor specializes Fl_Browser() by setting the type to FL←MULTI_BROWSER. The destructor destroys the widget and frees all memory that has been allocated.

The documentation for this class was generated from the following files:

- Fl_Multi_Browser.H
- Fl_Browser.cxx
**12.89 Fl_Multi_Label Struct Reference**

Allows a mixed text and/or graphics label to be applied to an `Fl_Menu_Item` or `Fl_Widget`.

```c
#include <Fl_Multi_Label.H>
```

**Public Member Functions**

- `void label(Fl_Menu_Item *)`
  
  Associate an `Fl_Multi_Label` with an `Fl_Menu_Item`.

- `void label(Fl_Widget *)`
  
  Associate an `Fl_Multi_Label` with an `Fl_Widget`.

**Public Attributes**

- `const char * labela`
  
  Holds the "leftmost" of the two elements in the composite label.

- `const char * labelb`
  
  Holds the "rightmost" of the two elements in the composite label.

- `uchar typea`
  
  Holds the "type" of labela.

- `uchar typeb`
  
  Holds the "type" of labelb.

**12.89.1 Detailed Description**

Allows a mixed text and/or graphics label to be applied to an `Fl_Menu_Item` or `Fl_Widget`. Most regular FLTK widgets now support the ability to associate both images and text with a label but some special cases, notably the non-widget `Fl_Menu_Item` objects, do not. `Fl_Multi_Label` may be used to create menu items that have an icon and text, which would not normally be possible for an `Fl_Menu_Item`. For example, `Fl_Multi_Label` is used in the New->Code submenu in fluid, and others.

![Menu items with icons using Fl_Multi_Label](image)

Each `Fl_Multi_Label` holds two elements, `labela` and `labelb`; each may hold either a text label (const char*) or an image (Fl_Image*). When displayed, `labela` is drawn first and `labelb` is drawn immediately to its right.

More complex labels can be constructed by setting `labelb` as another `Fl_Multi_Label` and thus chaining up a series of label elements.

When assigning a label element to one of `labela` or `labelb`, they should be explicitly cast to (const char*) if they are not of that type already.

**Note**

An `Fl_Multi_Label` object and all its components (label text, images, chained `Fl_Multi_Label`'s and their linked objects) must exist during the lifetime of the widget or menu item they are assigned to. It is the responsibility of the user's code to release these linked objects if necessary after the widget or menu is deleted.

**Example Use: Fl_Menu_Bar**

```c
Fl_Pixmap *image = new Fl_Pixmap(..); // image for menu item; any Fl_Image based widget
Fl_Menu_Bar *menu = new Fl_Menu_Bar(..); // can be any Fl_Menu_ oriented widget (Fl_Choice, Fl_Menu_Button..)

// Create a menu item
```

Generated by Doxygen
```c
int i = menu->add("File/New", ..);
Fl_Menu_Item *item = (Fl_Menu_Item*)(menu->menu()[i]);
// Create a multi label, assign it an image + text
Fl_Multi_Label *ml = new Fl_Multi_Label;
// Left side of label is an image
ml->typea = FL_IMAGE_LABEL;
ml->labela = (const char*)image; // any Fl_Image widget: Fl_Fixmap, Fl_PNG_Image, etc..
// Right side of label is label text
ml->typeb = FL_NORMAL_LABEL;
ml->labelb = item->label();
// Assign the multilabel to the menu item
// ml->label(item); // deprecated since 1.4.0; backwards compatible with 1.3.x
item->label(ml); // new method since 1.4.0
```

See also

Fl_Label and Fl_Labeltype and examples/howto-menu-with-images.cxx

### 12.89.2 Member Function Documentation

#### 12.89.2.1 `label()` [1/2]

```c
void Fl_Multi_Label::label (Fl_Menu_Item *o)
```

Associate an Fl_Multi_Label with an Fl_Menu_Item.

This uses Fl_Menu_Item::label(Fl_Labeltype a, const char *b) internally to set the label and the label type of the menu item, i.e. it stores a pointer to the Fl_Multi_Label object (this). An existing label (pointer) will be overwritten. This sets the type of the menu item's label to _FL_MULTI_LABEL - note the leading underscore ('_').

There is no way to use a method like Fl_Widget::copy_label() that transfers ownership of the Fl_Multi_Label and its linked objects (images, text, and chained Fl_Multi_Labels) to the menu item. The Fl_Multi_Label and all linked images, text labels, or chained Fl_Multi_Label objects must exist during the lifetime of the menu and will not be released when the menu item is destroyed.

Note

The user's code is responsible for releasing the Fl_Multi_Label and all linked objects (images, text, chained Fl_Multi_Label's) after the menu has been deleted. This may cause memory leaks if Fl_Multi_Label is used and reassigned w/o releasing the objects assigned to it.

**Deprecated** since 1.4.0: please use Fl_Menu_Item::label(Fl_Multi_Label *)

See also

Fl_Menu_Item::label(Fl_Multi_Label *)

#### 12.89.2.2 `label()` [2/2]

```c
void Fl_Multi_Label::label (Fl_Widget *o)
```

Associate an Fl_Multi_Label with an Fl_Widget.

This method uses Fl_Widget::label(Fl_Labeltype, const char *) internally to set the label of the widget, i.e. it stores a pointer to the Fl_Multi_Label object (this). An existing label that has been set using Fl_Widget::copy_label() will be released prior to the assignment of the new label.

This sets the type of the widget's label to _FL_MULTI_LABEL - note the leading underscore ('_').

There is no way to use a method like Fl_Widget::copy_label() that transfers ownership of the Fl_Multi_Label and its linked objects (images, text, and chained Fl_Multi_Labels) to the widget. The Fl_Multi_Label and all linked images, text labels, or chained Fl_Multi_Label objects must exist during the lifetime of the widget and will not be released when the widget is destroyed.

Note

The user's code is responsible for releasing the Fl_Multi_Label and all linked objects (images, text, chained Fl_Multi_Label's) after the widget has been deleted. This may cause memory leaks if Fl_Multi_Label is used and reassigned w/o releasing the objects assigned to it.
12.89.3 Member Data Documentation

12.89.3.1 labela

const char* Fl_Multi_Label::labela
Holds the "leftmost" of the two elements in the composite label.
Typically this would be assigned either a text string (const char*), a (Fl_Image*) or a (Fl_Multi_Label*).

12.89.3.2 labelb

const char* Fl_Multi_Label::labelb
Holds the "rightmost" of the two elements in the composite label.
Typically this would be assigned either a text string (const char*), a (Fl_Image*) or a (Fl_Multi_Label*).

12.89.3.3 typea

uchar Fl_Multi_Label::typea
Holds the "type" of labela.
Typically this is set to FL_NORMAL_LABEL for a text label, FL_IMAGE_LABEL for an image (based on Fl_image)
or FL_MULTI_LABEL if "chaining" multiple Fl_Multi_Label elements together.

12.89.3.4 typeb

uchar Fl_Multi_Label::typeb
Holds the "type" of labelb.
Typically this is set to FL_NORMAL_LABEL for a text label, FL_IMAGE_LABEL for an image (based on Fl_image)
or FL_MULTI_LABEL if "chaining" multiple Fl_Multi_Label elements together.
The documentation for this struct was generated from the following files:

• Fl_Multi_Label.H
• Fl_Multi_Label.cxx

12.90 Fl_Multiline_Input Class Reference

This input field displays 'n' characters as new lines rather than ^J, and accepts the Return, Tab, and up and down arrow keys.

#include <Fl_Multiline_Input.H>

Inheritance diagram for Fl_Multiline_Input:

Fl_Widget

Fl_Input_

Fl_Input

Fl_Multiline_Input

Public Member Functions

• Fl_Multiline_Input (int X, int Y, int W, int H, const char *=0)

  Creates a new Fl_Multiline_Input widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Input

• Fl_Input (int, int, int, int, const char *=0)
1118 Class Documentation

Creates a new Fl Input widget using the given position, size, and label string.

- **int handle (int) FL_OVERRIDE**
  Handles the specified event.

### Public Member Functions inherited from Fl_Input_

- **int append (const char *t, int l=0, char keep_selection=0)**
  Append text at the end.
- **bool can_redo () const**
  Check if there is a redo action available.
- **bool can_undo () const**
  Check if the last operation can be undone.
- **int copy (int clipboard)**
  Put the current selection into the clipboard.
- **int copy_cuts ()**
  Copies the yank buffer to the clipboard.
- **Fl_Color cursor_color () const**
  Gets the color of the cursor.
- **void cursor_color (Fl_Color n)**
  Sets the color of the cursor.
- **int cut ()**
  Deletes the current selection.
- **int cut (int a, int b)**
  Deletes all characters between index a and b.
- **int cut (int n)**
  Deletes the next n bytes rounded to characters before or after the cursor.
- **double dvalue () const**
  Returns the widget text interpreted as a floating point number.
- **Fl_Input_ (int, int, int, int, const char *=0)**
  Creates a new Fl_Input_ widget.
- **unsigned int index (int i) const**
  Returns the character at index i.
- **int input_type () const**
  Gets the input field type.
- **void input_type (int t)**
  Sets the input field type.
- **int insert (const char *t, int l=0)**
  Inserts text at the cursor position.
- **int insert_position () const**
  Gets the position of the text cursor.
- **int insert_position (int p)**
  Sets the cursor position and mark.
- **int insert_position (int p, int m)**
  Sets the index for the cursor and mark.
- **int ivalue () const**
  Returns the widget text interpreted as a signed integer.
- **int mark () const**
  Gets the current selection mark.
- **int mark (int m)**
  Sets the current selection mark.
- **int maximum_size () const**

Generated by Doxygen
Gets the maximum length of the input field in characters.

- void maximum_size (int m)
  Sets the maximum length of the input field in characters.

- int position () const
- int position (int p)
- int position (int p, int m)
- int readonly () const
  Gets the read-only state of the input field.

- void readonly (int b)
  Sets the read-only state of the input field.

- int redo ()
  Redo previous undo operation.

- int replace (int b, int e, const char *text, int ilen=0)
  Deletes text from b to e and inserts the new string text.

- void resize (int, int, int, int) FL_OVERRIDE
  Changes the size of the widget.

- int shortcut () const
  Return the shortcut key associated with this widget.

- void shortcut (int s)
  Sets the shortcut key associated with this widget.

- int size () const
  Returns the number of bytes in value().

- void size (int W, int H)
  Sets the width and height of this widget.

- int static_value (const char *)
  Changes the widget text.

- int static_value (const char *, int)
  Changes the widget text.

- int tab_nav () const
  Gets whether the Tab key causes focus navigation in multiline input fields or not.

- void tab_nav (int val)
  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.

- Fl_Color textcolor () const
  Gets the color of the text in the input field.

- void textcolor (Fl_Color n)
  Sets the color of the text in the input field.

- Fl_Font textfont () const
  Gets the font of the text in the input field.

- void textfont (Fl_Font s)
  Sets the font of the text in the input field.

- Fl_Fontsize textsize () const
  Gets the size of the text in the input field.

- void textsize (Fl_Fontsize s)
  Sets the size of the text in the input field.

- int undo ()
  Undoes previous changes to the text buffer.

- const char * value () const
  Returns the text displayed in the widget.

- int value (const char *)
  Changes the widget text.

- int value (const char *, int)
Changes the widget text.

- **int value (double value)**
  Changes the widget text to a floating point number ("%g").
- **int value (int value)**
  Changes the widget text to a signed integer number.
- **int wrap () const**
  Gets the word wrapping state of the input field.
- **void wrap (int b)**
  Sets the word wrapping state of the input field.
- **`Fl_Input_()`**
  Destroys the widget.

**Public Member Functions inherited from Fl_Widget**

- **void _clear_fullscreen ()**
- **void _set_fullscreen ()**
- **void activate ()**
  Activates the widget.
- **unsigned int active () const**
  Returns whether the widget is active.
- **int active_r () const**
  Returns whether the widget and all of its parents are active.
- **Fl_Align align () const**
  Gets the label alignment.
- **void align (Fl_Align alignment)**
  Sets the label alignment.
- **long argument () const**
  Gets the current user data (long) argument that is passed to the callback function.
- **void argument (long v)**
  Sets the current user data (long) argument that is passed to the callback function.
- **virtual class Fl_Gl_Window * as_gl_window ()**
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- **virtual class Fl_Gl_Window const * as_gl_window () const**
- **virtual Fl_Group * as_group ()**
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- **virtual Fl_Group const * as_group () const**
- **virtual Fl_Window * as_window ()**
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- **virtual Fl_Window const * as_window () const**
- **void bind_deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the inactive state.
- **void bind_deimage (int f)**
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- **void bind_image (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the active state.
- **void bind_image (int f)**
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- **Fl_Boxtype box () const**
  Gets the box type of the widget.
- **void box (Fl_Boxtype new_box)**
  Sets the box type for the widget.
• Fl_Callback_p callback () const
  Gets the current callback function for the widget.
• void callback (Fl_Callback ∗cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback ∗cb, void ∗p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 ∗cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 ∗cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
  Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget ∗w) const
  Checks if w is a child of this widget.
• void copy_label (const char ∗new_label)
  Sets the current label.
• void copy_tooltip (const char ∗text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int)
  Internal use only.
• void deactivate ()
Deactivates the widget.

- **Fl_Image** * deimage ()
  
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image** * deimage () const
  
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const
  
  Returns whether the inactive image is managed by the widget.

- **void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with default arguments.

- **void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with arbitrary arguments.

- **void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with arbitrary arguments.

- **void draw_label (int, int, int, int, Fl_Align) const
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h () const
  
  Gets the widget height.

- **virtual void hide ()
  
  Makes a widget invisible.

- **Fl_Image** * image ()
  
  Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image** * image () const
  
  Gets the image that is used as part of the widget label when in the active state.

- **void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the active state.

- **void image (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const
  
  Returns whether the image is managed by the widget.

- **int inside (const Fl_Widget *wgt) const
  
  Checks if this widget is a child of \textit{wgt}.

- **int is_label_copied () const
  
  Returns whether the current label was assigned with \textit{copy_label}().

- **const char * label () const
  
  Gets the current label text.

- **void label (const char *text)
  
  Sets the current label pointer.

- **void label (Fl_Labeltype a, const char *b)
  
  Shortcut to set the label text and type in one call.

- **Fl_Color labelcolor () const
  
  Gets the label color.

- **void labelcolor (Fl_Color c)
  
  Sets the label color.

- **Fl_Font labelfont () const
  
  Gets the font to use.

- **void labelfont (Fl_Font f)
  
  Sets the font to use.
• `Fl_Fontsize labelsize () const`
  Gets the font size in pixels.

• void `labelsize (Fl_Fontsize pix)`
  Sets the font size in pixels.

• `Fl_Labeltype labeltype () const`
  Gets the label type.

• void `labeltype (Fl_Labeltype a)`
  Sets the label type.

• void `measure_label (int &ww, int &hh) const`
  Sets width `ww` and height `hh` accordingly with the label size.

• bool `needs_keyboard () const`
  Returns whether this widget needs a keyboard.

• void `needs_keyboard (bool needs)`
  Sets whether this widget needs a keyboard.

• unsigned int `output () const`
  Returns if a widget is used for output only.

• `Fl_Group * parent () const`
  Returns a pointer to the parent widget.

• void `parent (Fl_Group * p)`
  Internal use only - “for hacks only”.

• void `position (int X, int Y)`
  Repositions the window or widget.

• void `redraw ()`
  Schedules the drawing of the widget.

• void `redraw_label ()`
  Schedules the drawing of the label.

• `Fl_Color selection_color () const`
  Gets the selection color.

• void `selection_color (Fl_Color a)`
  Sets the selection color.

• void `set_active ()`
  Marks the widget as active without sending events or changing focus.

• void `set_changed ()`
  Marks the value of the widget as changed.

• void `set_output ()`
  Sets a widget to output only.

• void `set_visible ()`
  Makes the widget visible.

• void `set_visible_focus ()`
  Enables keyboard focus navigation with this widget.

• int `shortcut_label () const`
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.

• void `shortcut_label (int value)`
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.

• virtual void `show ()`
  Makes a widget visible.

• void `size (int W, int H)`
  Changes the size of the widget.

• int `take_focus ()`
  Gives the widget the keyboard focus.

• unsigned int `takesevents () const`
• int test_shortcut ()
  Returns true if the widget’s label contains the entered ‘&x’ shortcut.

• const char * tooltip () const
  Gets the current tooltip text.

• void tooltip (const char *text)
  Sets the current tooltip text.

• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const
  Gets the widget type.

• void type (uchar t)
  Sets the widget type.

• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void * user_data () const
  Gets the user data for this widget.

• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

• void user_data (void *v)
  Sets the user data for this widget.

• unsigned int visible () const
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

• void visible_focus (int v)
  Modifies keyboard focus navigation.

• int visible_r () const
  Returns whether a widget and all its parents are visible.

• int w () const
  Gets the widget width.

• Fl_When when () const
  Returns the conditions under which the callback is called.

• void when (uchar i)
  Sets the flags used to decide when a callback is called.

• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  Gets the widget position in its window.

• int y () const
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  Destroys the widget.
Additional Inherited Members

Static Public Member Functions inherited from `Fl_Widget`

- static void `default_callback (Fl_Widget *widget, void *data)`
  *The default callback for all widgets that don’t set a callback.*

- static unsigned int `label_shortcut (const char *t)`
  *Returns the Unicode value of the ‘&x’ shortcut in a given text.*

- static int `test_shortcut (const char *t, const bool require_alt=false)`
  *Returns true if the given text t contains the entered ‘&x’ shortcut.*

Static Public Attributes inherited from `Fl_Input`

- static const char * `copy_menu_text` = "Copy"
  *[this text may be customized at run-time]*

- static const char * `cut_menu_text` = "Cut"
  *[this text may be customized at run-time]*

- static const char * `paste_menu_text` = "Paste"
  *[this text may be customized at run-time]*

Protected Types inherited from `Fl_Widget`

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 ,
  CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 ,
  CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 ,
  NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 ,
  MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  *flags possible values enumeration.*

Protected Member Functions inherited from `Fl_Input`

- void `draw () FL_OVERRIDE`
  *Draws the widget.*

- int `handle_key ()`
  *Handles a keystroke.*

- int `handle_rmb ()`
  *Handle right mouse button down events.*

Protected Member Functions inherited from `Fl_Input_`

- int `apply_undo ()`
  *Apply the current undo/redo operation.*

- void `drawtext (int, int, int, int)`
  *Draws the text in the passed bounding box.*

- void `drawtext (int, int, int, int, bool draw_active)`
  *Draws the text in the passed bounding box.*

- void `handle_mouse (int, int, int, int keepmark=0)`
Handles mouse clicks and mouse moves.

- int handletext (int e, int, int, int, int)
  Handles all kinds of text field related events.

- int line_end (int i) const
  Finds the end of a line.

- int line_start (int i) const
  Finds the start of a line.

- int linesPerPage ()
- void maybe_do_callback (FL_Callback_Reason reason=FL_REASON_UNKNOWN)
  Moves the cursor to the column given by up_down_pos.

- int word_end (int i) const
  Finds the end of a word.

- int word_start (int i) const
  Finds the start of a word.

- int xscroll () const
- int yscroll () const
- void yscroll (int yOffset)

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.

- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void draw_box () const
  Draws the widget box according its box style.

- void draw_box (FL_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

- void draw_box (FL_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- void draw_focus () const
  Draws a focus rectangle around the widget.

- void draw_focus (FL_Boxtype t, int X, int Y , int W, int H) const
  Draws a focus rectangle around the widget.

- void draw_focus (FL_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

- void draw_label () const
  Draws the widget's label at the defined label position.

- void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.

- unsigned int flags () const
  Gets the widget flags mask.

- void h (int v)
  Internal use only.

- void set_flag (unsigned int c)
  Sets a flag in the flags mask.

- void w (int v)
  Internal use only.
12.90.1 Detailed Description

This input field displays 'n' characters as new lines rather than ^J, and accepts the Return, Tab, and up and down arrow keys.
This is for editing multiline text.
This is far from the nirvana of text editors, and is probably only good for small bits of text, 10 lines at most. Note that this widget does not support scrollbars or per-character color control.
If you are presenting large amounts of text and need scrollbars or full color control of characters, you probably want Fl_Text_Editor instead.
In FLTK 1.3.x, the default behavior of the 'Tab' key was changed to support consistent focus navigation. To get the older FLTK 1.1.x behavior, set Fl_Input_::tab_nav() to 0. Newer programs should consider using Fl_Text_Editor.

12.90.2 Constructor & Destructor Documentation

12.90.2.1 Fl_Multiline_Input()

Fl_Multiline_Input::Fl_Multiline_Input (int X, int Y , int W, int H, const char ∗l=0 )

Creates a new Fl_Multiline_Input widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.
Inherited destructor destroys the widget and any value associated with it.
The documentation for this class was generated from the following files:
  • Fl_Multiline_Input.H
  • Fl_Input.cxx

12.91 Fl_Multiline_Output Class Reference

This widget is a subclass of Fl_Output that displays multiple lines of text.
#include <Fl_Multiline_Output.H>
Inheritance diagram for Fl_Multiline_Output:

Public Member Functions

  • Fl_Multiline_Output (int X, int Y, int W, int H, const char ∗l=0)
    Creates a new Fl_Multiline_Output widget using the given position, size, and label string.
Public Member Functions inherited from Fl_Output

- Fl_Output (int X, int Y, int W, int H, const char *l=0)
  Creates a new Fl_Output widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Input

- Fl_Input (int, int, int, int, const char *l=0)
  Creates a new Fl_Input widget using the given position, size, and label string.

- int handle (int) FL_OVERRIDE
  Handles the specified event.

Public Member Functions inherited from Fl_Input_

- int append (const char *t, int l=0, char keep_selection=0)
  Append text at the end.

- bool can_redo () const
  Check if there is a redo action available.

- bool can_undo () const
  Check if the last operation can be undone.

- int copy (int clipboard)
  Put the current selection into the clipboard.

- int copy_cuts ()
  Copies the yank buffer to the clipboard.

- Fl_Color cursor_color () const
  Gets the color of the cursor.

- void cursor_color (Fl_Color n)
  Sets the color of the cursor.

- int cut ()
  Deletes the current selection.

- int cut (int a, int b)
  Deletes all characters between index a and b.

- int cut (int n)
  Deletes the next n bytes rounded to characters before or after the cursor.

- double dvalue () const
  Returns the widget text interpreted as a floating point number.

- Fl_Input_ (int, int, int, int, const char *l=0)
  Creates a new Fl_Input_ widget.

- unsigned int index (int i) const
  Returns the character at index i.

- int input_type () const
  Gets the input field type.

- void input_type (int t)
  Sets the input field type.

- int insert (const char *t, int l=0)
  Inserts text at the cursor position.

- int insert_position () const
  Gets the position of the text cursor.

- int insert_position (int p)
  Sets the cursor position and mark.

- int insert_position (int p, int m)
  Sets the index for the cursor and mark.
• int ivalue () const
  Returns the widget text interpreted as a signed integer.

• int mark () const
  Gets the current selection mark.

• int mark (int m)
  Sets the current selection mark.

• int maximum_size () const
  Gets the maximum length of the input field in characters.

• void maximum_size (int m)
  Sets the maximum length of the input field in characters.

• int position () const
• int position (int p)
• int position (int p, int m)
  Gets the read-only state of the input field.

• void readonly (int b)
  Sets the read-only state of the input field.

• int redo ()
  Redo previous undo operation.

• int replace (int b, int e, const char ∗text, int ilen=0)
  Deletes text from b to e and inserts the new string text.

• void resize (int, int, int, int) FL_OVERRIDE
  Changes the size of the widget.

• int shortcut () const
  Return the shortcut key associated with this widget.

• void shortcut (int s)
  Sets the shortcut key associated with this widget.

• int size () const
  Returns the number of bytes in value().

• void size (int W, int H)
  Sets the width and height of this widget.

• int static_value (const char ∗)
  Changes the widget text.

• int static_value (const char ∗, int)
  Changes the widget text.

• int tab_nav () const
  Gets whether the Tab key causes focus navigation in multiline input fields or not.

• void tab_nav (int val)
  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.

• FL_COLOR textcolor () const
  Gets the color of the text in the input field.

• void textcolor (FL_COLOR n)
  Sets the color of the text in the input field.

• FL_FONT textfont () const
  Gets the font of the text in the input field.

• void textfont (FL_FONT s)
  Sets the font of the text in the input field.

• FL_FONTSIZE textsize () const
  Gets the size of the text in the input field.

• void textsize (FL_FONTSIZE s)
  Sets the size of the text in the input field.
• int undo ()
  Undoes previous changes to the text buffer.
• const char * value () const
  Returns the text displayed in the widget.
• int value (const char *)
  Changes the widget text.
• int value (const char *, int)
  Changes the widget text.
• int value (double value)
  Changes the widget text to a floating point number ("%g").
• int value (int value)
  Changes the widget text to a signed integer number.
• int wrap () const
  Gets the word wrapping state of the input field.
• void wrap (int b)
  Sets the word wrapping state of the input field.
• ~Fl_Input ()
  Destroys the widget.

Public Member Functions inherited from Fl_Widget
• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  Activates the widget.
• unsigned int active () const
  Returns whether the widget is active.
• int active_r () const
  Returns whether the widget and all of its parents are active.
• Fl_Align align () const
  Gets the label alignment.
• void align (Fl_Align alignment)
  Sets the label alignment.
• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
• virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const * as_gl_window () const
• virtual Fl_Group * as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• virtual Fl_Group const * as_group () const
• virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const * as_window () const
• void bind_deimage (Fl_Image * img)
  Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image (Fl_Image * img)
Sets the image to use as part of the widget label when in the active state.

- **void bind_image (int f)**
  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype box () const**
  
  Gets the box type of the widget.

- **void box (Fl_Boxtype new_box)**
  
  Sets the box type for the widget.

- **Fl_Callback_p callback () const**
  
  Gets the current callback function for the widget.

- **void callback (Fl_Callback *cb)**
  
  Sets the current callback function for the widget.

- **void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)**
  
  Sets the current callback function and managed user data for the widget.

- **void callback (Fl_Callback *cb, void *p)**
  
  Sets the current callback function and data for the widget.

- **void callback (Fl_Callback0 *cb)**
  
  Sets the current callback function for the widget.

- **void callback (Fl_Callback1 *cb, long p=0)**
  
  Sets the current callback function for the widget.

- **unsigned int changed () const**
  
  Checks if the widget value changed since the last callback.

- **void clear_active ()**
  
  Marks the widget as inactive without sending events or changing focus.

- **void clear_changed ()**
  
  Marks the value of the widget as unchanged.

- **void clear_damage (uchar c=0)**
  
  Clears or sets the damage flags.

- **void clear_output ()**
  
  Sets a widget to accept input.

- **void clear_visible ()**
  
  Hides the widget.

- **void clear_visible_focus ()**
  
  Disables keyboard focus navigation with this widget.

- **Fl_Color color () const**
  
  Gets the background color of the widget.

- **void color (Fl_Color bg)**
  
  Sets the background color of the widget.

- **void color (Fl_Color bg, Fl_Color sel)**
  
  Sets the background and selection color of the widget.

- **Fl_Color color2 () const**
  
  For back compatibility only.

- **void color2 (unsigned a)**
  
  For back compatibility only.

- **int contains (const Fl_Widget *w) const**
  
  Checks if w is a child of this widget.

- **void copy_label (const char *new_label)**
  
  Sets the current label.

- **void copy_tooltip (const char *text)**
  
  Sets the current tooltip text.

- **uchar damage () const**
  
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image ∗ deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image ∗ deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget ∗ widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget ∗ widget, void ∗ arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image ∗ image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image ∗ image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget ∗ wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char ∗ label () const
  Gets the current label text.
• void label (const char ∗ text)
  Sets the current label text.
• Fl_Color labelcolor () const
  Shortcut to set the label text and type in one call.
Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.

• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.

• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.

• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.

• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.

• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.

• unsigned int output () const
  Returns if a widget is used for output only.

• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - “for hacks only”.

• void position (int X, int Y)
  Repositions the window or widget.

• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.

• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.

• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.

• void set_output ()
  Sets a widget to output only.

• void set_visible ()
  Makes the widget visible.

• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.

• int shortcut_label () const
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.
• virtual void show ()
   
   Makes a widget visible.
• void size (int W, int H)
   
   Changes the size of the widget.
• int take_focus ()
   
   Gives the widget the keyboard focus.
• unsigned int takeevents () const
   
   Returns if the widget is able to take events.
• int test_shortcut ()
   
   Returns true if the widget's label contains the entered '&x' shortcut.
• const char * tooltip () const
   
   Gets the current tooltip text.
• void tooltip (const char *text)
   
   Sets the current tooltip text.
• Fl_Window * top_window () const
   
   Returns a pointer to the top-level window for the widget.
• Fl_Window * top_window_offset (int &xoff, int &yoff) const
   
   Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
   
   Gets the widget type.
• void type (uchar t)
   
   Sets the widget type.
• int use_accents_menu ()
   
   Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * user_data () const
   
   Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
   
   Sets the user data for this widget.
• void user_data (void *v)
   
   Sets the user data for this widget.
• unsigned int visible () const
   
   Returns whether a widget is visible.
• unsigned int visible_focus () const
   
   Checks whether this widget has a visible focus.
• void visible_focus (int v)
   
   Modifies keyboard focus navigation.
• int visible_r () const
   
   Returns whether a widget and all its parents are visible.
• int w () const
   
   Gets the widget width.
• Fl_When when () const
   
   Returns the conditions under which the callback is called.
• void when (uchar i)
   
   Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
   
   Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
   
   Gets the widget position in its window.
• int y () const
   
   Gets the widget position in its window.
• virtual ~Fl_Widget ()
   
   Destroys the widget.
Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char ∗t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char ∗t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Static Public Attributes inherited from Fl_Input

- static const char ∗copy_menu_text = "Copy"
  [this text may be customized at run-time]
- static const char ∗cut_menu_text = "Cut"
  [this text may be customized at run-time]
- static const char ∗paste_menu_text = "Paste"
  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLBOX_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLBOX = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Input

- void draw () FL_OVERRIDE
  Draws the widget.
- int handle_key ()
  Handles a keystroke.
- int handle_rmb ()
  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

- int apply_undo ()
  Apply the current undo/redo operation.
- void drawtext (int, int, int, int)
  Draws the text in the passed bounding box.
- void drawtext (int, int, int, int, bool draw_active)
  Draws the text in the passed bounding box.
- void handle_mouse (int, int, int, int keepmark=0)

Generated by Doxygen
Handles mouse clicks and mouse moves.

- int handleText (int e, int, int, int, int)
  Handles all kinds of text field related events.

- int line_end (int i) const
  Finds the end of a line.

- int line_start (int i) const
  Finds the start of a line.

- int linesPerPage ()

- void maybe_do_callback (FL_Callback_Reason reason=FL_REASON_UNKNOWN)
  Moves the cursor to the column given by up_down_pos.

- int word_end (int i) const
  Finds the end of a word.

- int word_start (int i) const
  Finds the start of a word.

- int xscroll () const
- int yscroll () const
- void yscroll (int yOffset)

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.

- void draw_backdrop () const
  Draws the widget box according its box style.

- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- void draw_focus () const
  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

- void draw_label () const
  Draws the widget's label at the defined label position.

- void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.

- unsigned int flags () const
  Gets the widget flags mask.

- void h (int v)
  Internal use only.

- void set_flag (unsigned int c)
  Sets a flag in the flags mask.

- void w (int v)
  Internal use only.
12.91.1 Detailed Description

This widget is a subclass of Fl_Output that displays multiple lines of text. It also displays tab characters as whitespace to the next column. Note that this widget does not support scrollbars, or per-character color control. If you are presenting large amounts of read-only text and need scrollbars, or full color control of characters, then use Fl_Text_Display. If you want to display HTML text, use Fl_Help_View.

A caret cursor (\textasciitilde) shows the keyboard navigation mark for keyboard selection of the output text, e.g. Arrow Keys to move the cursor, Shift + Arrow Keys to create a text selection, and \textasciitilde' to copy the selected text to the paste buffer. The caret cursor can be disabled by disabling the widget’s “visible focus” using clear_visible_focus(), inherited from the Fl_Widget base class. Doing this also disables the widget’s keyboard navigation.

12.91.2 Constructor & Destructor Documentation

12.91.2.1 Fl_Multiline_Output()

Fl_Multiline_Output::Fl_Multiline_Output (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char ∗ l = 0 )

Creates a new Fl_Multiline_Output widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX.

Inherited destructor destroys the widget and any value associated with it.

The documentation for this class was generated from the following files:

- Fl_Multiline_Output.H
- Fl_Multiline_Outputxx

12.92 Fl_Native_File_Chooser Class Reference

This class lets an FLTK application easily and consistently access the operating system's native file chooser.

\#include <Fl_Native_File_Chooser.H>

Public Types

- enum Option {
  NO_OPTIONS = 0x0000 , SAVEAS_CONFIRM = 0x0001 , NEW_FOLDER = 0x0002 , PREVIEW = 0x0004 ,
  USE_FILTER_EXT = 0x0008 }
- enum Type {
  BROWSE_FILE = 0 , BROWSE_DIRECTORY , BROWSE_MULTI_FILE , BROWSE_MULTI_DIRECTORY ,
  BROWSE_SAVE_FILE , BROWSE_SAVE_DIRECTORY }

Public Member Functions

- int count () const
  
  Returns the number of filenames (or directory names) the user selected.

- const char ∗ directory () const
  
  Returns the current preset directory() value.

- void directory (const char ∗val)
  
  Preset the directory the browser will show when opened.
• const char * errmsg () const
  
  *Returns a system dependent error message for the last method that failed.*

• const char * filename () const

  *Return the filename the user chose.*

• const char * filename (int i) const

  *Return one of the filenames the user selected.*

• const char * filter () const

  *Returns the filter string last set.*

• void filter (const char * f)

  *Sets the filename filters used for browsing.*

• int filter_value () const

  *Returns which filter value was last selected by the user.*

• void filter_value (int i)

  *Sets which filter will be initially selected.*

• int filters () const

  *Gets how many filters were available, not including "All Files".*

• Fl_Native_File_Chooser (int val=BROWSE_FILE)

  *The constructor.*

• int options () const

  *Gets the platform specific Fl_Native_File_Chooser::Option flags.*

• void options (int o)

  *Sets the platform specific chooser options to val.*

• const char * preset_file () const

  *Get the preset filename.*

• void preset_file (const char * f)

  *Sets the default filename for the chooser.*

• int show ()

  *Post the chooser's dialog.*

• const char * title () const

  *Get the title of the file chooser's dialog window.*

• void title (const char * t)

  *Set the title of the file chooser's dialog window.*

• int type () const

  *Gets the current Fl_Native_File_Chooser::Type of browser.*

• void type (int t)

  *Sets the current Fl_Native_File_Chooser::Type of browser.*

• ~Fl_Native_File_Chooser ()

  *Destructor.*

**Static Public Attributes**

• static const char * file_exists_message = "File exists. Are you sure you want to overwrite?"

  *Localizable message.*
12.92 Fl_Native_File_Chooser Class Reference

12.92.1 Detailed Description

This class lets an FLTK application easily and consistently access the operating system's native file chooser. Some operating systems have very complex and specific file choosers that many users want access to specifically, instead of FLTK's default file chooser(s).

In cases where there is no native file browser, FLTK's own file browser is used instead.

To use this widget, use the following include in your code:

```
#include <FL/Fl_Native_File_Chooser.H>
```

The following example shows how to pick a single file:

```
// Create and post the local native file chooser
#include <FL/Fl_Native_File_Chooser.H>
[..]
Fl_Native_File_Chooser fnfc;
fnfc.title("Pick a file");
fnfc.type(Fl_Native_File_Chooser::BROWSE_FILE);
fnfc.filter("Text\n*.txt\n";  
"C Files\t*.cxx,h,c");
fnfc.directory("/var/tmp"); // default directory to use
// Show native chooser
switch ( fnfc.show() ) {  
  case -1: printf("ERROR: %s\n", fnfc.errmsg()); break; // ERROR
  case 1: printf("CANCEL\n"); break; // CANCEL
  default: printf("PICKED: %s\n", fnfc.filename()); break; // FILE CHOSEN
}
```

The `Fl_Native_File_Chooser` widget transmits UTF-8 encoded filenames to its user. It is recommended to open files that may have non-ASCII names with the `fl_fopen()` or `fl_open()` utility functions that handle these names in a cross-platform way (whereas the standard fopen()/open() functions fail on the Windows platform to open files with a non-ASCII name).

Platform Specific Caveats

- Under X11/Wayland the dialog is chosen as follows:
  1. If command `zenity` is available at run-time and if Fl::option(OPTION_FNFCUSES_ZENITY) is not turned off, the zenity -based dialog opens. This is expected to be more appropriate than other dialog forms for sandboxed apps.
  2. Else if the app runs under the KDE desktop and if command `kdialog` is available at run-time and if the library was not built with `cmake -DOPTION_USE_KDIALOG=Off`, the `kdialog` -based dialog opens.
  3. Else if the GTK library is available at run-time on the computer and if Fl::option(OPTION_FNFCUSES_GTK) is not turned off, the GTK-styled dialog opens. Call `fl_register_images()` to add a "Preview" button to this dialog. Use the static public attributes of class `Fl_File_Chooser` to localize the browser.
  4. Otherwise, FLTK's own dialog produced by the `Fl_File_Chooser` class opens. Call `fl_register_images()` to add a "Preview" button to it. It's best if you also call `Fl_File_Icon::load_system_icons()` at the start of `main()`, to enable the nicer looking file browser widgets. Use the static public attributes of class `Fl_File_Chooser` to localize the browser.

Todo Improve documentation about selection of native file choosers on X11/Wayland.

- Some operating systems support certain OS specific options; see `Fl_Native_File_Chooser::options()` for a list.
12.92.2 Member Enumeration Documentation

12.92.2.1 Option

```cpp
enum Fl_Native_File_Chooser::Option
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_OPTIONS</td>
<td>no options enabled</td>
</tr>
<tr>
<td>SAVEAS_CONFIRM</td>
<td>Show native 'Save As' overwrite confirm dialog.</td>
</tr>
<tr>
<td>NEW_FOLDER</td>
<td>Show 'New Folder' icon (if supported)</td>
</tr>
<tr>
<td>PREVIEW</td>
<td>enable preview mode (if supported)</td>
</tr>
<tr>
<td>USE_FILTER_EXT</td>
<td>Chooser filter presets the output file extension (if supported)</td>
</tr>
</tbody>
</table>

12.92.2.2 Type

```cpp
enum Fl_Native_File_Chooser::Type
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROWSE_FILE</td>
<td>browse files (lets user choose one file)</td>
</tr>
<tr>
<td>BROWSE_DIRECTORY</td>
<td>browse directories (lets user choose one directory)</td>
</tr>
<tr>
<td>BROWSE_MULTI_FILE</td>
<td>browse files (lets user choose multiple files)</td>
</tr>
<tr>
<td>BROWSE_MULTI_DIRECTORY</td>
<td>browse directories (lets user choose multiple directories)</td>
</tr>
<tr>
<td>BROWSE_SAVE_FILE</td>
<td>browse to save a file</td>
</tr>
<tr>
<td>BROWSE_SAVE_DIRECTORY</td>
<td>browse to save a directory</td>
</tr>
</tbody>
</table>
12.92.3 Constructor & Destructor Documentation

12.92.3.1 Fl_Native_File_Chooser()

Fl_Native_File_Chooser::Fl_Native_File_Chooser (  
    int val = BROWSE_FILE  )

The constructor.
Internally allocates the native widgets. Optional \texttt{val} presets the type of browser this will be, which can also be changed with \texttt{type()}.

12.92.3.2 \texttt{\sim \ }Fl_Native_File_Chooser()

Fl_Native_File_Chooser::\texttt{\sim \ }Fl_Native/File_Chooser ( )
Destructor.
Deallocates any resources allocated to this widget.

12.92.4 Member Function Documentation

12.92.4.1 count()

\begin{verbatim}
int Fl_Native_File_Chooser::count ( ) const
\end{verbatim}

Returns the number of filenames (or directory names) the user selected.
Example:
\begin{verbatim}
if ( fnfc->show() == 0 ) {  
    // Print all filenames user selected  
    for (int n=0; n<fnfc->count(); n++) {  
        printf("%d) '%s'
", n, fnfc->filename(n));  
    }  
}
\end{verbatim}

12.92.4.2 directory()

\begin{verbatim}
void Fl_Native_File_Chooser::directory (  
    const char * val  )
\end{verbatim}

Preset the directory the browser will show when opened.
If \texttt{val} is \texttt{NULL}, or no directory is specified, the chooser will attempt to use the last non-cancelled folder.

12.92.4.3 errmsg()

\begin{verbatim}
const char * Fl_Native/File_Chooser::errmsg ( ) const
\end{verbatim}

Returns a system dependent error message for the last method that failed.
This message should at least be flagged to the user in a dialog box, or to some kind of error log. Contents will be valid only for methods that document \texttt{errmsg()} will have info on failures.

12.92.4.4 filename() [1/2]

\begin{verbatim}
const char * Fl_Native_File_Chooser::filename ( ) const
\end{verbatim}

Return the filename the user chose.
Use this if only expecting a single filename. If more than one filename is expected, use \texttt{filename(int)} instead. Return value may be "" if no filename was chosen (eg. user cancelled).

12.92.4.5 filename() [2/2]

\begin{verbatim}
const char * Fl_Native_File_Chooser::filename (  
    int i  ) const
\end{verbatim}

Return one of the filenames the user selected.
Use \texttt{count()} to determine how many filenames the user selected.
Example:
\begin{verbatim}
if ( fnfc->show() == 0 ) {  
    // Print all filenames user selected  
    for (int n=0; n<fnfc->count(); n++) {  
        printf("%d) '%s'
", n, fnfc->filename(n));  
    }  
}
\end{verbatim}
12.92.4.6  filter() [1/2]

const char * Fl_Native_File_Chooser::filter ( ) const
Returns the filter string last set.
Can be NULL if no filter was set.

12.92.4.7  filter() [2/2]

void Fl_Native_File_Chooser::filter ( const char * f )
Sets the filename filters used for browsing.
The default is NULL, which browses all files.
The filter string can be any of:

• A single wildcard (eg. "*.txt")
• Multiple wildcards (eg. "*.cxx,h,H")
• A descriptive name followed by a "t" and a wildcard (eg. "Text Files\t.txt")
• A list of separate wildcards with a "n" between each (eg. "*.cxx,h,H\n*.txt")
• A list of descriptive names and wildcards (eg. "C++ Files\t*.cxx,h,H\nTxt Files\t*.txt")

The format of each filter is a wildcard, or an optional user description followed by 't' and the wildcard.

12.92.4.8  filter_value() [1/2]

int Fl_Native_File_Chooser::filter_value ( ) const
Returns which filter value was last selected by the user.
This is only valid if the chooser returns success and if the particular file chooser supports it. Otherwise the value is not changed.
Some "native" file choosers don't support returning the filter selection by the user, particularly the system dialog based browsers:

• kdialog (KDE system dialog)
• zenity (another system dialog based chooser).

Note: this list may not be complete.
These system file chooser dialogs don't return the filter value chosen by the user.
See also

filter(const char *f)

12.92.4.9  filter_value() [2/2]

void Fl_Native_File_Chooser::filter_value ( int i )
Sets which filter will be initially selected.
The first filter is indexed as 0. If filter_value() == filters(), then "All Files" was chosen. If filter_value() > filters(), then a custom filter was set.
Some “native” file choosers don't support this way to set the initial filter type, particularly the system dialog based browsers:

• kdialog (KDE system dialog)
• zenity (another system dialog based chooser).

Note: this list may not be complete.
As far as we know these dialogs use the first item in the list of filter values as the initial filter presented to the user.
12.92 Fl_Native_File_Chooser Class Reference

See also

```
filter(const char *f)
```

### 12.92.4.10 options()

```cpp
void Fl_Native_File_Chooser::options (int o)
```

Sets the platform specific chooser options to `val`. `val` is expected to be one or more `Fl_Native_File_Chooser::Option` flags ORed together. Some platforms have OS-specific functions that can be enabled/disabled via this method.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Win</th>
<th>Mac</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW_FOLDER</td>
<td>Shows the 'New Folder' button.</td>
<td>Ignored</td>
<td>Used</td>
<td>Used</td>
</tr>
<tr>
<td>PREVIEW</td>
<td>Enables the 'Preview' mode by default.</td>
<td>Ignored</td>
<td>Ignored</td>
<td>Used</td>
</tr>
<tr>
<td>SAVESAS_CONFIRM</td>
<td>Confirm dialog if BROWSE_SAVE_FILE file exists.</td>
<td>Used</td>
<td>Used</td>
<td>Used</td>
</tr>
<tr>
<td>USE_FILTER_EXT</td>
<td>Chooser filter presets the output file extension.</td>
<td>Ignored</td>
<td>Used (GTK)</td>
<td></td>
</tr>
</tbody>
</table>

### 12.92.4.11 preset_file()

```cpp
void Fl_Native_File_Chooser::preset_file (const char *f)
```

Sets the default filename for the chooser. Use `directory()` to set the default directory. Mainly used to preset the filename for save dialogs, and on most platforms can be used for opening files as well.

### 12.92.4.12 show()

```cpp
int Fl_Native_File_Chooser::show ()
```

Post the chooser's dialog. Blocks until dialog has been completed or cancelled.

Returns

- 0 – user picked a file
- 1 – user cancelled
- -1 – failed; errmsg() has reason

### 12.92.4.13 title() [1/2]

```cpp
const char * Fl_Native_File_Chooser::title ( ) const
```

Get the title of the file chooser's dialog window. Return value may be NULL if no title was set.

### 12.92.4.14 title() [2/2]

```cpp
void Fl_Native_File_Chooser::title (const char * t)
```

Set the title of the file chooser's dialog window. Can be NULL if no title desired. The default title varies according to the platform, so you are advised to set the title explicitly.

The documentation for this class was generated from the following files:

- `Fl_Native_File_Chooser.H`
- `Fl_Native_File_Chooser.cxx`
- `Fl_Native_File_Chooser_GTK.cxx`
12.93 **Fl_Nice_Slider Class Reference**

Inheritance diagram for Fl_Nice_Slider:

```
Fl_Widget
   ↓
Fl_Valuator
   ↓
Fl_Slider
   ↓
Fl_Nice_Slider
```

**Public Member Functions**

- **Fl_Nice_Slider** (int X, int Y, int W, int H, const char ∗L=0)

**Public Member Functions inherited from Fl_Slider**

- void *bounds* (double a, double b)
  
  _Sets the minimum (a) and maximum (b) values for the valuator widget._

- **Fl_Slider** (int X, int Y, int W, int H, const char ∗L=0)
  
  _Creates a new Fl_Slider widget using the given position, size, and label string._

- **Fl_Slider** (uchar t, int X, int Y, int W, int H, const char ∗L)
  
  _Creates a new Fl_Slider widget using the given type, position, size, and label string._

- int *handle* (int) FL_OVERRIDE
  
  _Handles the specified event._

- int *scrollvalue* (int pos, int size, int first, int total)
  
  _Sets the size and position of the sliding knob in the box._

- **Fl_Boxtype slider() const**
  
  _Gets the slider box type._

- void *slider* (Fl_Boxtype c)
  
  _Sets the slider box type._

- float *slider_size() const
  
  _Get the dimensions of the moving piece of slider._

- void *slider_size* (double v)
  
  _Set the dimensions of the moving piece of slider._

**Public Member Functions inherited from Fl_Valuator**

- void *bounds* (double a, double b)
  
  _Sets the minimum (a) and maximum (b) values for the valuator widget._

- double *clamp* (double)
  
  _Clamps the passed value to the valuator range._

- virtual int *format* (char ∗)
  
  _Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter._

- double *increment* (double, int)
  
  _Adds n times the step value to the passed value._

- double *maximum* () const
  
  _Gets the maximum value for the valuator._

- void *maximum* (double a)
Sets the maximum value for the valuator.

- double minimum() const
  
  Gets the minimum value for the valuator.

- void minimum(double a)
  
  Sets the minimum value for the valuator.

- void precision(int digits)
  
  Sets the step value to \(1.0 / 10^{\text{digits}}\).

- void range(double a, double b)
  
  Sets the minimum and maximum values for the valuator.

- double round(double)
  
  Round the passed value to the nearest step increment.

- double step() const
  
  Gets or sets the step value.

- void step(double a, int b)
  
  See double Fl_Valuator::step() const

- void step(double s)
  
  See double Fl_Valuator::step() const.

- void step(int a)
  
  See double Fl_Valuator::step() const

- double value() const
  
  Gets the floating point(double) value.

- int value(double)
  
  Sets the current value.

### Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen()
- void _set_fullscreen()
- void activate()

  Activates the widget.

- unsigned int active() const
  
  Returns whether the widget is active.

- int active_r() const
  
  Returns whether the widget and all of its parents are active.

- Fl_Align align() const
  
  Gets the label alignment.

- void align(Fl_Align alignment)
  
  Sets the label alignment.

- long argument() const
  
  Gets the current user data (long) argument that is passed to the callback function.

- void argument(long v)

  Sets the current user data (long) argument that is passed to the callback function.

- virtual class Fl_Gl_Window as_gl_window()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const as_gl_window() const

- virtual Fl_Group as_group()

  Returns an Fl_Group pointer if this widget is an Fl_Group.

- virtual Fl_Group const as_group() const

- virtual Fl_Window as_window()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
virtual Fl_Window const * as_window () const

Sets the image to use as part of the widget label when in the inactive state.

void bind_deimage (Fl_Image *img)

Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

void bind_image (int f)

Sets the image to use as part of the widget label when in the active state.

void bind_image (Fl_Image *img)

Bind the image to the widget, so the widget will delete the image when it is no longer needed.

Fl_Boxtype box () const

Gets the box type of the widget.

void box (Fl_Boxtype new_box)

Sets the box type for the widget.

Fl_Callback_p callback () const

Gets the current callback function for the widget.

void callback (Fl_Callback *cb)

Sets the current callback function for the widget.

void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

Sets the current callback function and managed user data for the widget.

void callback (Fl_Callback *cb, void *p)

Sets the current callback function and data for the widget.

void callback (Fl_Callback0 *cb)

Sets the current callback function for the widget.

void callback (Fl_Callback1 *cb, long p=0)

Sets the current callback function for the widget.

unsigned int changed () const

Checks if the widget value changed since the last callback.

void clear_active ()

Marks the widget as inactive without sending events or changing focus.

void clear_changed ()

Marks the value of the widget as unchanged.

void clear_damage (uchar c=0)

Clears or sets the damage flags.

void clear_output ()

Sets a widget to accept input.

void clear_visible ()

Hides the widget.

void clear_visible_focus ()

Disables keyboard focus navigation with this widget.

Fl_Color color () const

Gets the background color of the widget.

void color (Fl_Color bg)

Sets the background color of the widget.

void color (Fl_Color bg, Fl_Color sel)

Sets the background and selection color of the widget.

Fl_Color color2 () const

For back compatibility only.

void color2 (unsigned a)

For back compatibility only.

int contains (const Fl_Widget *w) const

Checks if w is a child of this widget.
• void copy_label (const char *new_label)
  Sets the current label.
• void copy_tooltip (const char *text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image * deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image * deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char * label () const
Gets the current label text.

- **void label (const char *text)**
  Sets the current label pointer.

- **void label (Fl_Labeltype a, const char *b)**
  Shortcut to set the label text and type in one call.

**Fl_Color labelcolor () const**

- Gets the label color.

**void labelcolor (Fl_Color c)**

- Sets the label color.

**Fl_Font labelfont () const**

- Gets the font to use.

**void labelfont (Fl_Font f)**

- Sets the font to use.

**Fl_Fontsize labelsize () const**

- Gets the font size in pixels.

**void labelsize (Fl_Fontsize pix)**

- Sets the font size in pixels.

**Fl_Labeltype labeltype () const**

- Gets the label type.

**void labeltype (Fl_Labeltype a)**

- Sets the label type.

**void measure_label (int &ww, int &hh) const**

- Sets width ww and height hh accordingly with the label size.

**bool needs_keyboard () const**

- Returns whether this widget needs a keyboard.

**void needs_keyboard (bool needs)**

- Sets whether this widget needs a keyboard.

**unsigned int output () const**

- Returns if a widget is used for output only.

**Fl_Group *parent () const**

- Returns a pointer to the parent widget.

**void parent (Fl_Group *p)**

- Internal use only - "for hacks only".

**void position (int X, int Y)**

- Repositions the window or widget.

**void redraw ()**

- Schedules the drawing of the widget.

**void redraw_label ()**

- Schedules the drawing of the label.

**virtual void resize (int x, int y, int w, int h)**

- Changes the size or position of the widget.

**Fl_Color selection_color () const**

- Gets the selection color.

**void selection_color (Fl_Color a)**

- Sets the selection color.

**void set_active ()**

- Marks the widget as active without sending events or changing focus.

**void set_changed ()**

- Marks the value of the widget as changed.

**void set_output ()**

- Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses ' & ' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses ' & ' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered ' & x ' shortcut.
• const char ∗ tooltip () const
  Gets the current tooltip text.
• void tooltip (const char ∗text)
  Sets the current tooltip text.
• Fl_Window ∗ top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void ∗ user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void ∗v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
Sets the flags used to decide when a callback is called.

- **FL_Window** + window () const
  
  Returns a pointer to the nearest parent window up the widget hierarchy.

- int x () const
  
  Gets the widget position in its window.

- int y () const
  
  Gets the widget position in its window.

- virtual ~Fl_Widget ()
  
  Destroys the widget.

**Additional Inherited Members**

**Static Public Member Functions inherited from Fl_Widget**

- static void default_callback (Fl_Widget *widget, void *data)
  
  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)
  
  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)
  
  Returns true if the given text *t* contains the entered '&x' shortcut.

**Protected Types inherited from Fl_Widget**

- enum {

  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31
  }

  flags possible values enumeration.

**Protected Member Functions inherited from Fl_Slider**

- void draw () FL_OVERRIDE
  
  Draws the widget.

- void draw (int, int, int, int)

- int handle (int, int, int, int)

**Protected Member Functions inherited from Fl_Valuator**

- **Fl_Valuator** (int X, int Y , int W, int H, const char *L)
  
  Creates a new Fl_Valuator widget using the given position, size, and label string.

- void handle_drag (double newvalue)
  
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.

- void handle_push ()
  
  Stores the current value in the previous value.

- void handle_release ()
  
  Called after an FL_WHEN_RELEASE event is received and before the callback.
• int horizontal() const
  Tells if the valuator is an FL_HORIZONTAL one.
• double previous_value() const
  Gets the previous floating point value before an event changed it.
• void set_value(double v)
  Sets the current floating point value.
• double softclamp(double)
  Clamps the value, but accepts v if the previous value is not already out of range.
• virtual void value_damage()
  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

• void clear_flag(unsigned int c)
  Clears a flag in the flags mask.
• void draw_backdrop() const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box() const
  Draws the widget box according its box style.
• void draw_box(Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
• void draw_box(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
• void draw_focus() const
  Draws a focus rectangle around the widget.
• void draw_focus(Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
• void draw_focus(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
• void draw_label() const
  Draws the widget's label at the defined label position.
• void draw_label(int, int, int, int) const
  Draws the label in an arbitrary bounding box.
• Fl_Widget(int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.
• unsigned int flags() const
  Gets the widget flags mask.
• void h(int v)
  Internal use only.
• void set_flag(unsigned int c)
  Sets a flag in the flags mask.
• void w(int v)
  Internal use only.
• void x(int v)
  Internal use only.
• void y(int v)
  Internal use only.

The documentation for this class was generated from the following files:

• Fl_Nice_Slider.H
• Fl_Slider.cxx

Generated by Doxygen
12.94  Fl_Output Class Reference

This widget displays a piece of text.

```cpp
#include <Fl_Output.H>
```

Inheritance diagram for Fl_Output:

![Inheritance Diagram]

Public Member Functions

- **Fl_Output** (int X, int Y, int W, int H, const char *l=0)
  
  Creates a new Fl_Output widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Input

- **Fl_Input** (int, int, int, int, const char *l=0)
  
  Creates a new Fl_Input widget using the given position, size, and label string.

- int **handle** (int) FL_OVERRIDE
  
  Handles the specified event.

Public Member Functions inherited from Fl_Input_

- int **append** (const char *t, int l=0, char keep_selection=0)
  
  Append text at the end.

- bool **can_redo** () const
  
  Check if there is a redo action available.

- bool **can_undo** () const
  
  Check if the last operation can be undone.

- int **copy** (int clipboard)
  
  Put the current selection into the clipboard.

- int **copy_cuts** ()
  
  Copies the yank buffer to the clipboard.

- Fl_Color **cursor_color** () const
  
  Gets the color of the cursor.

- void **cursor_color** (Fl_Color n)
  
  Sets the color of the cursor.

- int **cut** ()
  
  Deletes the current selection.

- int **cut** (int a, int b)
  
  Deletes all characters between index a and b.

- int **cut** (int n)
  
  Deletes the next n bytes rounded to characters before or after the cursor.

- double **dvalue** () const
Returns the widget text interpreted as a floating point number.

• **Fl_Input_** (int, int, int, int, const char *=0)

  Creates a new Fl_Input_ widget.

• unsigned int **index** (int i) const

  Returns the character at index i.

• int **input_type** () const

  Gets the input field type.

• void **input_type** (int t)

  Sets the input field type.

• int **insert** (const char *=t, int l=0)

  Inserts text at the cursor position.

• int **insert_position** () const

  Gets the position of the text cursor.

• int **insert_position** (int p)

  Sets the cursor position and mark.

• int **insert_position** (int p, int m)

  Sets the index for the cursor and mark.

• int **ivalue** () const

  Returns the widget text interpreted as a signed integer.

• int **mark** () const

  Gets the current selection mark.

• int **mark** (int m)

  Sets the current selection mark.

• int **maximum_size** () const

  Gets the maximum length of the input field in characters.

• void **maximum_size** (int m)

  Sets the maximum length of the input field in characters.

• int **position** () const

• int **position** (int p)

• int **position** (int p, int m)

• int **readonly** () const

  Gets the read-only state of the input field.

• void **readonly** (int b)

  Sets the read-only state of the input field.

• int **redo** ()

  Redo previous undo operation.

• int **replace** (int b, int e, const char *=text, int ilen=0)

  Deletes text from b to e and inserts the new string text.

• void **resize** (int, int, int, int) FL_OVERRIDE

  Changes the size of the widget.

• int **shortcut** () const

  Return the shortcut key associated with this widget.

• void **shortcut** (int s)

  Sets the shortcut key associated with this widget.

• int **size** () const

  Returns the number of bytes in value().

• void **size** (int W, int H)

  Sets the width and height of this widget.

• int **static_value** (const char *=)

  Changes the widget text.

• int **static_value** (const char *=, int)
Changes the widget text.

• int tab_nav () const
  Gets whether the Tab key causes focus navigation in multiline input fields or not.

• void tab_nav (int val)
  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.

• Fl_Color textcolor () const
  Gets the color of the text in the input field.

• void textcolor (Fl_Color n)
  Sets the color of the text in the input field.

• Fl_Font textfont () const
  Gets the font of the text in the input field.

• void textfont (Fl_Font s)
  Sets the font of the text in the input field.

• Fl_Fontsize textsize () const
  Gets the size of the text in the input field.

• void textsize (Fl_Fontsize s)
  Sets the size of the text in the input field.

• int undo ()
  Undoes previous changes to the text buffer.

• const char * value () const
  Returns the text displayed in the widget.

• int value (const char * value)
  Changes the widget text.

• int value (const char *, int)
  Changes the widget text.

• int value (double value)
  Changes the widget text to a floating point number ("%g").

• int value (int value)
  Changes the widget text to a signed integer number.

• int wrap () const
  Gets the word wrapping state of the input field.

• void wrap (int b)
  Sets the word wrapping state of the input field.

• ~Fl_Input_ ()
  Destroys the widget.

Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  Activates the widget.
• unsigned int active () const
  Returns whether the widget is active.

• int active_r () const
  Returns whether the widget and all of its parents are active.

• Fl_Align align () const
  Gets the label alignment.

• void align (Fl_Align alignment)
  Sets the label alignment.
• long argument () const
• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const * as_gl_window () const

• virtual Fl_Group * as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Group const * as_group () const

• virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window const * as_window () const

• void bind_deimage (Fl_Image * img)
  Sets the image to use as part of the widget label when in the inactive state.

• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

• void bind_image (Fl_Image * img)
  Sets the image to use as part of the widget label when in the active state.

• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

• Fl_Boxtype box () const
  Gets the box type of the widget.

• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.

• Fl_Callback_p callback () const
  Gets the current callback function for the widget.

• void callback (Fl_Callback * cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback * cb, Fl_Callback_User_Data * p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

• void callback (Fl_Callback * cb, void * p)
  Sets the current callback function and data for the widget.

• void callback (Fl_Callback0 * cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback1 * cb, long p=0)
  Sets the current callback function for the widget.

• unsigned int changed () const
  Checks if the widget value changed since the last callback.

• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

• void clear_changed ()
  Marks the value of the widget as unchanged.

• void clear_damage (uchar c=0)
  Clears or sets the damage flags.

• void clear_output ()
  Sets a widget to accept input.

• void clear_visible ()
  Hides the widget.

• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

• Fl_Color color () const
  Returns the color of the widget.
Gets the background color of the widget.

- void color (Fl_Color bg)
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

- Fl_Color color2 () const
  For back compatibility only.

- void color2 (unsigned a)
  For back compatibility only.

- int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.

- void copy_label (const char *new_label)
  Sets the current label.

- void copy_tooltip (const char *text)
  Sets the current tooltip text.

- uchar damage () const
  Returns non-zero if draw() needs to be called.

- void damage (uchar c)
  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- int damage_resize (int, int, int, int)
  Internal use only.

- void deactivate ()
  Deactivates the widget.

- Fl_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

- const Fl_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

- void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

- void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_CbReason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

- void do_callback (Fl_Widget *witness, long arg, Fl_CbReason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- void do_callback (Fl_Widget *witness, void *arg=0, Fl_CbReason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- void draw_label (int, int, int, int, Fl_Align)
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- int h () const
  Gets the widget height.

- virtual void hide ()
  Makes a widget invisible.

- Fl_Image *image ()
  Gets the image that is used as part of the widget label when in the active state.

- const Fl_Image *image () const
  Gets the image that is used as part of the widget label when in the active state.
• **void image (Fl_Image &img)**
  
  Sets the image to use as part of the widget label when in the active state.

• **void image (Fl_Image *img)**
  
  Sets the image to use as part of the widget label when in the active state.

• **int image_bound () const**

  Returns whether the image is managed by the widget.

• **int inside (const Fl_Widget *wgt) const**

  Checks if this widget is a child of wgt.

• **int is_label_copied () const**

  Returns whether the current label was assigned with copy_label().

• **const char * label () const**

  Gets the current label text.

• **void label (const char *text)**

  Sets the current label pointer.

• **void label (Fl_Labeltype a, const char *b)**

  Shortcut to set the label text and type in one call.

• **Fl_Color labelcolor () const**

  Gets the label color.

• **void labelcolor (Fl_Color c)**

  Sets the label color.

• **Fl_Font labelfont () const**

  Gets the font to use.

• **void labelfont (Fl_Font f)**

  Sets the font to use.

• **Fl_Fontsize labelsize () const**

  Gets the font size in pixels.

• **void labelsize (Fl_Fontsize pix)**

  Sets the font size in pixels.

• **Fl_Labeltype labeltype () const**

  Gets the label type.

• **void labeltype (Fl_Labeltype a)**

  Sets the label type.

• **void measure_label (int &ww, int &hh) const**

  Sets width ww and height hh accordingly with the label size.

• **bool needs_keyboard () const**

  Returns whether this widget needs a keyboard.

• **void needs_keyboard (bool needs)**

  Sets whether this widget needs a keyboard.

• **unsigned int output () const**

  Returns if a widget is used for output only.

• **Fl_Group * parent () const**

  Returns a pointer to the parent widget.

• **void parent (Fl_Group *p)**

  Internal use only - “for hacks only”.

• **void position (int X, int Y)**

  Repositions the window or widget.

• **void redraw ()**

  Schedules the drawing of the widget.

• **void redraw_label ()**

  Schedules the drawing of the label.

• **Fl_Color selection_color () const**

  Returns the color used for selection.
• void \texttt{selection\_color (Fl\_Color a)}
  
  \textit{Sets the selection color.}

• void \texttt{set\_active ()}
  
  \textit{Marks the widget as active without sending events or changing focus.}

• void \texttt{set\_changed ()}
  
  \textit{Marks the value of the widget as changed.}

• void \texttt{set\_output ()}
  
  \textit{Sets a widget to output only.}

• void \texttt{set\_visible ()}
  
  \textit{Makes the widget visible.}

• void \texttt{set\_visible\_focus ()}
  \begin{quote}
  \textit{Enables keyboard focus navigation with this widget.}
  \end{quote}

• int \texttt{shortcut\_label () const}
  
  \textit{Returns whether the widget's label uses ' & ' to indicate shortcuts.}

• void \texttt{shortcut\_label (int value)}
  
  \textit{Sets whether the widget's label uses ' & ' to indicate shortcuts.}

• virtual void \texttt{show ()}
  
  \textit{Makes a widget visible.}

• void \texttt{size (int W, int H)}
  
  \textit{Changes the size of the widget.}

• int \texttt{take\_focus ()}
  
  \textit{Gives the widget the keyboard focus.}

• unsigned int \texttt{take\_events () const}
  
  \textit{Returns if the widget is able to take events.}

• int \texttt{test\_shortcut ()}
  
  \textit{Returns true if the widget's label contains the entered ' & x ' shortcut.}

• const char * \texttt{tooltip () const}
  
  \textit{Gets the current tooltip text.}

• void \texttt{tooltip (const char *text)}
  
  \textit{Sets the current tooltip text.}

• \texttt{Fl\_Window * top\_window () const}
  
  \textit{Returns a pointer to the top-level window for the widget.}

• \texttt{Fl\_Window * top\_window\_offset (int &xoff, int &yoff) const}
  
  \textit{Finds the x/y offset of the current widget relative to the top-level window.}

• uchar \texttt{type () const}
  
  \textit{Gets the widget type.}

• void \texttt{type (uchar t)}
  
  \textit{Sets the widget type.}

• int \texttt{use\_accents\_menu ()}
  
  \textit{Returns non zero if MAC\_USE\_ACCENTS\_MENU flag is set, 0 otherwise.}

• void * \texttt{user\_data () const}
  
  \textit{Gets the user data for this widget.}

• void \texttt{user\_data (Fl\_Callback\_User\_Data *v, bool auto\_free)}
  
  \textit{Sets the user data for this widget.}

• void \texttt{user\_data (void *v)}
  
  \textit{Sets the user data for this widget.}

• unsigned int \texttt{visible () const}
  
  \textit{Returns whether a widget is visible.}

• unsigned int \texttt{visible\_focus () const}
  
  \textit{Checks whether this widget has a visible focus.}
- void visible_focus (int v)
  Modifies keyboard focus navigation.
- int visible_r () const
  Returns whether a widget and all its parents are visible.
- int w () const
  Gets the widget width.
- Fl_When when () const
  Returns the conditions under which the callback is called.
- void when (uchar i)
  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const
  Gets the widget position in its window.
- int y () const
  Gets the widget position in its window.
- virtual ~Fl_Widget ()
  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Static Public Attributes inherited from Fl_Input

- static const char * copy_menu_text = "Copy"
  [this text may be customized at run-time]
- static const char * cut_menu_text = "Cut"
  [this text may be customized at run-time]
- static const char * paste_menu_text = "Paste"
  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.
Protected Member Functions inherited from Fl_Input

• void draw () FL_OVERRIDE
  Draws the widget.
• int handle_key ()
  Handles a keystroke.
• int handle_rmb ()
  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

• int apply_undo ()
  Apply the current undo/redo operation.
• void drawtext (int, int, int, int)
  Draws the text in the passed bounding box.
• void drawtext (int, int, int, int, bool draw_active)
  Draws the text in the passed bounding box.
• void handle_mouse (int, int, int, int keepmark=0)
  Handles mouse clicks and mouse moves.
• int handletext (int e, int, int, int, int)
  Handles all kinds of text field related events.
• int line_end (int i) const
  Finds the end of a line.
• int line_start (int i) const
  Finds the start of a line.
• int linesPerPage ()
• void maybe_do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
• int up_down_position (int, int keepmark=0)
  Moves the cursor to the column given by up_down_pos.
• int word_end (int i) const
  Finds the end of a word.
• int word_start (int i) const
  Finds the start of a word.
• int xscroll () const
• int yscroll () const
• void yscroll (int yOffset)

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const
  Draws the widget box according its box style.
• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
• void draw_focus () const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

- void draw_label () const
  Draws the widget's label at the defined label position.

- void draw_label (int, int, int) const
  Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.

- unsigned int flags () const
  Gets the widget flags mask.

- void h (int v)
  Internal use only.

- void set_flag (unsigned int c)
  Sets a flag in the flags mask.

- void w (int v)
  Internal use only.

- void x (int v)
  Internal use only.

- void y (int v)
  Internal use only.

### 12.94.1 Detailed Description

This widget displays a piece of text. When you set the value(), Fl_Output does a strcpy() to its own storage, which is useful for program-generated values. The user may select portions of the text using the mouse and paste the contents into other fields or programs.

![Fl_Output Example](image)

There is a single subclass, Fl_Multiline_Output, which allows you to display multiple lines of text. Fl_Multiline_Output does not provide scroll bars. If a more complete text editing widget is needed, use Fl_Text_Display instead.

The text may contain any characters except \0, and will correctly display anything, using \^X notation for unprintable control characters and \nnn notation for unprintable characters with the high bit set. It assumes the font can draw any characters in the ISO-Latin1 character set.
12.94.2 Constructor & Destructor Documentation

12.94.2.1 Fl_Output()

Fl_Output::Fl_Output (int X, int Y, int W, int H, const char ∗l = 0)

Creates a new Fl_Output widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX. Inherited destructor destroys the widget and any value associated with it. The documentation for this class was generated from the following files:

• Fl_Output.H
• Fl_Input.cxx

12.95 Fl_Overlay_Window Class Reference

This window provides double buffering and also the ability to draw the "overlay" which is another picture placed on top of the main image.

#include <Fl_Overlay_Window.H>

Inheritance diagram for Fl_Overlay_Window:

```
Fl_Widget
  Fl_Group
  Fl_Window
  Fl_Double_Window
  Fl_Overlay_Window
```

Public Member Functions

• Fl_Overlay_Window ∗ as_overlay_window () FL_OVERRIDE
  Return non-null if this is an Fl_Overlay_Window object.

• int can_do_overlay ()
  Returns non-zero if there's hardware overlay support.

• virtual void draw_overlay ()=0
  You must subclass Fl_Overlay_Window and provide this method.

• void flush () FL_OVERRIDE
  Forces the window to be drawn, this window is also made current and calls draw().

• void hide () FL_OVERRID
  Makes a widget invisible.

• void redraw_overlay ()
  Call this to indicate that the overlay data has changed and needs to be redrawn.

• void resize (int, int, int, int) FL_OVERRID
  Changes the size or position of the widget.

• void show () FL_OVERRID
  Makes a widget visible.
• void **show** (int a, char ∗∗b)
  
  Same as *Fl_Window::show(int a, char ∗∗b)*

• **~Fl_Overlay_Window**
  
  Destroys the window and all child widgets.

### Public Member Functions inherited from Fl_Double_Window

• Fl_Double_Window *as_double_window () FL_OVERRIDE
  
  Return non-null if this is an Fl_Overlay_Window object.

• Fl_Double_Window (int W, int H, const char ∗=0)
  
  Creates a new Fl_Double_Window widget using the given position, size, and label (title) string.

• Fl_Double_Window (int X, int Y, int W, int H, const char ∗=0)
  
  See *Fl_Double_Window::Fl_Double_Window(int w, int h, const char ∗label = 0)*

• void **show** (int a, char ∗∗b)
  
  Same as *Fl_Window::show(int a, char ∗∗b)*

• **~Fl_Double_Window**
  
  The destructor also deletes all the children.

### Public Member Functions inherited from Fl_Window

• Fl_Window const *as_window () const FL_OVERRIDE

• Fl_Window ∗as_window () FL_OVERRIDE
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• unsigned int **border** () const
  
  Returns whether the window possesses a border.

• void **border** (int b)
  
  Sets whether or not the window manager border is around the window.

• void **clear_border** ()
  
  Fast inline function to turn the window manager border off.

• void **clear_modal_states** ()
  
  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.

• void **copy_label** (const char ∗a)
  
  Sets the window titlebar label to a copy of a character string.

• void **cursor** (const Fl_RGB_Image ∗, int, int)
  
  Changes the cursor for this window using the provided image as cursor's shape.

• void **cursor** (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  
  For back compatibility only.

• void **cursor** (Fl_Cursor)
  
  Changes the cursor for this window.

• int **decorated_h** () const
  
  Returns the window height including any window title bar and any frame added by the window manager.

• int **decorated_w** () const
  
  Returns the window width including any frame added by the window manager.

• void **default_cursor** (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  
  For back compatibility only.

• void **default_cursor** (Fl_Cursor)
  
  Sets the default window cursor.

• void **draw_backdrop** ()
  
  Draw the background image if one is set and is aligned inside.

• Fl_Window (int w, int h, const char ∗title=0)
  
  Creates a window from the given width w, height h, and title.

• Fl_Window (int x, int y, int w, int h, const char ∗title=0)
Creates a window from the given position (x, y), size (w, h) and title.

- **void free_position ()**
  
  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.

- **void fullscreen ()**
  
  Makes the window completely fill one or more screens, without any window manager border visible.

- **unsigned int fullscreen_active () const**
  
  Returns non zero if FULLSCREEN flag is set, 0 otherwise.

- **void fullscreen_off ()**
  
  Turns off any side effects of fullscreen()

- **void fullscreen_off (int X, int Y, int W, int H)**
  
  Turns off any side effects of fullscreen() and does resize(x,y,w,h).

- **void fullscreen_screens (int top, int bottom, int left, int right)**
  
  Sets which screens should be used when this window is in fullscreen mode.

- **int handle (int) FL_OVERRIDE**
  
  Handles the specified event.

- **void hide () FL_OVERRIDE**
  
  Removes the window from the screen.

- **void hotspot (const Fl_Widget &p, int offscreen=0)**
  
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

- **void hotspot (const Fl_Widget *, int offscreen=0)**
  
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

- **void hotspot (int x, int y, int offscreen=0)**
  
  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.

- **const void * icon () const**
  
  Gets the current icon window target dependent data.

- **void icon (const Fl_RGB_Image *)**
  
  Sets or resets a single window icon.

- **void icon (const void *ic)**
  
  Platform-specific method to set the window icon usable on Windows and X11 only.

- **void iconize ()**
  
  Iconifies the window.

- **const char * iconlabel () const**
  
  See void Fl_Window::iconlabel(const char *)

- **void iconlabel (const char *)**
  
  Sets the icon label.

- **void icons (const Fl_RGB_Image *[])**, int)
  
  Sets the window icons.

- **void icons (HICON big_icon, HICON small_icon)**
  
  Sets the window icons using HICON handles (Windows platform only).

- **const char * label () const**
  
  See void Fl_Window::label(const char *)

- **void label (const char *)**
  
  Sets the window title bar label.

- **void label (const char *label, const char *iconlabel)**
  
  Sets the icon label.

- **void make_current ()**
  
  Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.
• void maximize ()
  Maximizes a top-level window to its current screen.
• unsigned int maximize_active () const
  Returns whether the window is currently maximized.
• unsigned int menu_window () const
  Returns true if this window is a menu window.
• unsigned int modal () const
  Returns true if this window is modal.
• unsigned int non_modal () const
  Returns true if this window is modal or non-modal.
• fl_uintptr_t os_id ()
  Returns a platform-specific identification of a shown window, or 0 if not shown.
• unsigned int override () const
  Returns non zero if OVERRIDE flag is set, 0 otherwise.
• void resize (int X, int Y, int W, int H) FL_OVERRIDE
  Changes the size and position of the window.
• int screen_num ()
  The number of the screen containing the mapped window.
• void screen_num (int screen_num)
  Set the number of the screen where to map the window.
• void set_menu_window ()
  Marks the window as a menu window.
• void set_modal ()
  A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).
• void set_non_modal ()
  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.
• void set_override ()
  Activates the flags NOBORDER/OVERRIDE.
• void set_tooltip_window ()
  Marks the window as a tooltip window.
• const Fl_Image * shape ()
  Returns the image controlling the window shape or NULL.
• void shape (const Fl_Image &b)
  Set the window's shape with an Fl_Image.
• void shape (const Fl_Image *img)
  Assigns a non-rectangular shape to the window.
• void show () FL_OVERRIDE
  Puts the window on the screen.
• void show (int argc, char **argv)
  Puts the window on the screen with show() and parses command-line arguments.
• int shown ()
  Returns non-zero if show() has been called (but not hide() ).
• void size_range (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)
  Sets the allowable range the user can resize this window to.
• unsigned int tooltip_window () const
  Returns true if this window is a tooltip window.
• void un_maximize ()
  Returns a previously maximized top-level window to its previous size.
• void wait_for_expose ()
Waits for the window to be displayed after calling show().

- **int x_root () const**
  
  Gets the x position of the window on the screen.

- **const char * xclass () const**
  
  Returns the xclass for this window, or a default.

- **void xclass (const char * c)**
  
  Sets the xclass for this window.

- **int y_root () const**
  
  Gets the y position of the window on the screen.

- **virtual ~Fl_Window ()**
  
  The destructor also deletes all the children.

### Public Member Functions inherited from Fl_Group

- **Fl_Widget *& _ddfdesign_kludge ()**
  
  This is for forms compatibility only.

- **void add (Fl_Widget &)**
  
  The widget is removed from its current group (if any) and then added to the end of this group.

- **void add (Fl_Widget *o)**
  
  See void Fl_Group::add(Fl_Widget &w)

- **void add_resizable (Fl_Widget &o)**
  
  Adds a widget to the group and makes it the resizable widget.

- **Fl_Widget *const * array () const**
  
  Returns a pointer to the array of children.

- **Fl_Group const * as_group () const FL_OVERRIDE**

- **Fl_Group * as_group () FL_OVERRIDE**

  Returns an Fl_Group pointer if this widget is an Fl_Group.

- **void begin ()**
  
  Sets the current group so you can build the widget tree by just constructing the widgets.

- **Fl_Widget * child (int n) const**
  
  Returns array()[n].

- **int children () const**
  
  Returns how many child widgets the group has.

- **void clear ()**
  
  Deletes all child widgets from memory recursively.

- **unsigned int clip_children ()**
  
  Returns the current clipping mode.

- **void clip_children (int c)**
  
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- **virtual int delete_child (int n)**
  
  Removes the widget at index from the group and deletes it.

- **void end ()**
  
  Exactly the same as current(this->parent()).

- **int find (const Fl_Widget &o) const**
  
  See int Fl_Group::find(const Fl_Widget *w) const.

- **int find (const Fl_Widget *) const**
  
  Searches the child array for the widget and returns the index.

- **Fl_Group (int, int, int, int, const char * =0)**
  
  Creates a new Fl_Group widget using the given position, size, and label string.

- **void focus (Fl_Widget *)**

- **void forms_end ()**
This is for forms compatibility only.

- void init_sizes()
  Resets the internal array of widget sizes and positions.
- void insert(Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.
- void insert(Fl_Widget &o, Fl_Widget *before)
  This does insert(w, find(before)).
- void remove(Fl_Widget &)
  Removes a widget from the group but does not delete it.
- void remove(Fl_Widget *o)
  Removes the widget o from the group.
- void remove(int index)
  Removes the widget at index from the group but does not delete it.
- Fl_Widget *resizable() const
  Returns the group's resizable widget.
- void resizable(Fl_Widget &o)
  Sets the group's resizable widget.
- void resizable(Fl_Widget *o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group()
  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen()
- void _set_fullscreen()
- void activate()
  Activates the widget.
- unsigned int active() const
  Returns whether the widget is active.
- int active_r() const
  Returns whether the widget and all of its parents are active.
- Fl_Align align() const
  Gets the label alignment.
- void align(Fl_Align alignment)
  Sets the label alignment.
- long argument() const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument(long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window *as_gl_window()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const *as_gl_window() const
- void bind_deimage(Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage(int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image(Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image(int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• **Fl_Boxtype box () const**
  
  Gets the box type of the widget.

• **void box (Fl_Boxtype new_box)**
  
  Sets the box type for the widget.

• **Fl_Callback_p callback () const**
  
  Gets the current callback function for the widget.

• **void callback (Fl_Callback +cb)**
  
  Sets the current callback function for the widget.

• **void callback (Fl_Callback +cb, Fl_Callback_User_Data +p, bool auto_free)**
  
  Sets the current callback function and managed user data for the widget.

• **void callback (Fl_Callback +cb, void +p)**
  
  Sets the current callback function and data for the widget.

• **void callback (Fl_Callback0 +cb)**
  
  Sets the current callback function for the widget.

• **void callback (Fl_Callback1 +cb, long p=0)**
  
  Sets the current callback function for the widget.

• **unsigned int changed () const**
  
  Checks if the widget value changed since the last callback.

• **void clear_active ()**
  
  Marks the widget as inactive without sending events or changing focus.

• **void clear_changed ()**
  
  Marks the value of the widget as unchanged.

• **void clear_damage (uchar c=0)**
  
  Clears or sets the damage flags.

• **void clear_output ()**
  
  Sets a widget to accept input.

• **void clear_visible ()**
  
  Hides the widget.

• **void clear_visible_focus ()**
  
  Disables keyboard focus navigation with this widget.

• **Fl_Color color () const**
  
  Gets the background color of the widget.

• **void color (Fl_Color bg)**
  
  Sets the background color of the widget.

• **void color (Fl_Color bg, Fl_Color sel)**
  
  Sets the background and selection color of the widget.

• **Fl_Color color2 () const**
  
  For back compatibility only.

• **void color2 (unsigned a)**
  
  For back compatibility only.

• **int contains (const Fl_Widget +w) const**
  
  Checks if w is a child of this widget.

• **void copy_label (const char +new_label)**
  
  Sets the current label.

• **void copy_tooltip (const char +text)**
  
  Sets the current tooltip text.

• **uchar damage () const**
  
  Returns non-zero if draw() needs to be called.

• **void damage (uchar c)**
  
  Sets the damage bits for the widget.

• **void damage (uchar c, int x, int y, int w, int h)**
Sets the damage bits for an area inside the widget.

- `int damage_resize(int, int, int, int)`  
  Internal use only.

- `void deactivate()`  
  Deactivates the widget.

- `FL_Image * deimage()`  
  Gets the image that is used as part of the widget label when in the inactive state.

- `const FL_Image * deimage()` const  
  Gets the image that is used as part of the widget label when in the inactive state.

- `void deimage(FL_Image &img)`  
  Sets the image to use as part of the widget label when in the inactive state.

- `void deimage(FL_Image *img)`  
  Sets the image to use as part of the widget label when in the inactive state.

- `int deimage_bound()` const  
  Returns whether the inactive image is managed by the widget.

- `void do_callback(FL_Callback_Reason reason=FL_REASON_UNKNOWN)`  
  Calls the widget callback function with default arguments.

- `void do_callback(FL_Widget *widget, long arg, FL_Callback_Reason reason=FL_REASON_UNKNOWN)`  
  Calls the widget callback function with arbitrary arguments.

- `void do_callback(FL_Widget *widget, void *arg=0, FL_Callback_Reason reason=FL_REASON_UNKNOWN)`  
  Calls the widget callback function with arbitrary arguments.

- `void draw_label(int, int, int, int, Fl_Align) const`  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- `int h()` const  
  Gets the widget height.

- `FL_Image * image()`  
  Gets the image that is used as part of the widget label when in the active state.

- `const FL_Image * image()` const  
  Gets the image that is used as part of the widget label when in the active state.

- `void image(FL_Image &img)`  
  Sets the image to use as part of the widget label when in the active state.

- `void image(FL_Image *img)`  
  Sets the image to use as part of the widget label when in the active state.

- `int image_bound()` const  
  Returns whether the image is managed by the widget.

- `int inside(const FL_Widget *wgt)` const  
  Checks if this widget is a child of `wgt`.

- `int is_label_copied()` const  
  Returns whether the current label was assigned with `copy_label()`.

- `const char * label()` const  
  Gets the current label text.

- `void label(const char *text)`  
  Sets the current label pointer.

- `void label(FL_Labeltype a, const char *b)`  
  Shortcut to set the label text and type in one call.

- `FL_Color labelcolor()` const  
  Gets the label color.

- `void labelcolor(FL_Color c)`  
  Sets the label color.

- `FL_Font labelfont()` const  
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
Returns if the widget is able to take events.

- int testShortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.

- const char * tooltip () const
  Gets the current tooltip text.

- void tooltip (const char *text)
  Sets the current tooltip text.

- Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.

- Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const
  Gets the widget type.

- void type (uchar t)
  Sets the widget type.

- int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void * user_data () const
  Gets the user data for this widget.

- void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

- void user_data (void *v)
  Sets the user data for this widget.

- unsigned int visible () const
  Returns whether a widget is visible.

- unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

- void visible_focus (int v)
  Modifies keyboard focus navigation.

- int visible_r () const
  Returns whether a widget and all its parents are visible.

- int w () const
  Gets the widget width.

- Fl_When when () const
  Returns the conditions under which the callback is called.

- void when (uchar i)
  Sets the flags used to decide when a callback is called.

- Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

- int x () const
  Gets the widget position in its window.

- int y () const
  Gets the widget position in its window.

- virtual ~Fl_Widget ()
  Destroys the widget.

Protected Member Functions

- Fl_Overlay_Window (int W, int H, const char *l=0)
  See Fl_Overlay_Window::Fl_Overlay_Window(int X, int Y, int W, int H, const char *l=0)

- Fl_Overlay_Window (int X, int Y, int W, int H, const char *l=0)
  Creates a new Fl_Overlay_Window widget using the given position, size, and label (title) string.
Protected Member Functions inherited from Fl_Window

- void default_size_range ()
  Protected method to calculate the default size range of a window.
- void draw () FL_OVERRIDE
  Draws the widget.
- int force_position () const
  Returns the internal state of the window's FORCE_POSITION flag.
- void force_position (int force)
  Sets an internal flag that tells FLTK and the window manager to honor position requests.
- void free_icons ()
  Deletes all icons previously attached to the window.
- int is_resizable ()
  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.
- void draw_children ()
  Draws all children of the group.
- void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)
  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()
  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget’s position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
• void **draw_focus** (**Fl_Boxtype** t, int x, int y, int w, int h, **Fl_Color** bg) const
  
  Draws a focus box for the widget at the given position and size.

• void **draw_label** () const

  Draws the widget's label at the defined label position.

• void **draw_label** (int, int, int, int) const

  Draws the label in an arbitrary bounding box.

• **Fl_Widget**(int x, int y, int w, int h, const char ∗ label=0L)

  Creates a widget at the given position and size.

• unsigned int **flags** () const

  Gets the widget flags mask.

• void **h**(int v)

  Internal use only.

• void **set_flag**(unsigned int c)

  Sets a flag in the flags mask.

• void **w**(int v)

  Internal use only.

• void **x**(int v)

  Internal use only.

• void **y**(int v)

  Internal use only.

**Additional Inherited Members**

**Public Types inherited from Fl_Window**

• typedef struct HICON__ ∗ HICON

**Static Public Member Functions inherited from Fl_Window**

• static Fl_Window ∗ **current** ()

  Returns the last window that was made current.

• static void **default_callback**(Fl_Window ∗, void ∗v)

  Back compatibility: Sets the default callback v for win to call on close event.

• static void **default_icon**(const Fl_RGB_Image ∗)

  Sets a single default window icon.

• static void **default_icons**(const Fl_RGB_Image ∗[], int)

  Sets the default window icons.

• static void **default_icons**(HICON big_icon, HICON small_icon)

  Sets the default window icons (Windows platform only).

• static const char ∗ **default_xclass** ()

  Returns the default xclass.

• static void **default_xclass**(const char ∗)

  Sets the default window xclass.

• static bool **is_a_rescale** ()

  Returns true when a window is being rescaled.

• static char **show_next_window_iconic** ()

  Returns the static flag whether the next window should be opened iconified.

• static void **show_next_window_iconic**(char stat)

  Sets a static flag whether the next window should be opened iconified.
Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()
  Returns the currently active group.
- static void current (Fl_Group *g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACENTS_MENU
  = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
flags possible values enumeration.

Static Protected Attributes inherited from Fl_Window

- static Fl_Window * current_
  Stores the last window that was made current.

12.95.1 Detailed Description

This window provides double buffering and also the ability to draw the "overlay" which is another picture placed on top of the main image.
The overlay is designed to be a rapidly-changing but simple graphic such as a mouse selection box. Fl_Overlay_Window uses the overlay planes provided by your graphics hardware if they are available.
If no hardware support is found the overlay is simulated by drawing directly into the on-screen copy of the double-buffered window, and "erased" by copying the backbuffer over it again. This means the overlay will blink if you change the image in the window.

12.95.2 Constructor & Destructor Documentation

12.95.2.1 Fl_Overlay_Window()

Fl_Overlay_Window::Fl_Overlay_Window ( int X, int Y, int W, )
Generated by Doxygen
int H,
const char * l = 0 ) [protected]
Creates a new Fl_Overlay_Window widget using the given position, size, and label (title) string.
If the positions (x,y) are not given, then the window manager will choose them.

12.95.3 Member Function Documentation

12.95.3.1 as_overlay_window()

Fl_Overlay_Window * Fl_Overlay_Window::as_overlay_window ( ) [inline], [virtual]
Return non-null if this is an Fl_Overlay_Window object.
Reimplemented from Fl_Window.

12.95.3.2 draw_overlay()

virtual void Fl_Overlay_Window::draw_overlay ( ) [pure virtual]
You must subclass Fl_Overlay_Window and provide this method.
It is just like a draw() method, except it draws the overlay. The overlay will have already been "cleared" when this is
called. You can use any of the routines described in <FL/fl_draw.H>.

12.95.3.3 flush()

void Fl_Overlay_Window::flush ( ) [virtual]
Forces the window to be drawn, this window is also made current and calls draw().
Reimplemented from Fl_Double_Window.

12.95.3.4 hide()

void Fl_Overlay_Window::hide ( ) [virtual]
Makes a widget invisible.
See also
    show(), visible(), visible_r()
Reimplemented from Fl_Double_Window.

12.95.3.5 redraw_overlay()

void Fl_Overlay_Window::redraw_overlay ( )
Call this to indicate that the overlay data has changed and needs to be redrawn.
The overlay will be clear until the first time this is called, so if you want an initial display you must call this after calling
show().

12.95.3.6 resize()

void Fl_Overlay_Window::resize ( int x, int y, int w, int h ) [virtual]
Changes the size or position of the widget.
This is a virtual function so that the widget may implement its own handling of resizing. The default version does
not call the redraw() method, but instead relies on the parent widget to do so because the parent may know a faster
way to update the display, such as scrolling from the old position.
Some window managers under X11 call resize() a lot more often than needed. Please verify that the position or
size of a widget did actually change before doing any extensive calculations.
position(X, Y) is a shortcut for resize(X, Y, w(), h()), and size(W, H) is a shortcut for resize(x(), y(), W, H).

Generated by Doxygen
Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y</td>
</tr>
<tr>
<td>in</td>
<td>w,h</td>
</tr>
</tbody>
</table>

See also

position(int,int), size(int,int)

Reimplemented from Fl_Double_Window.

12.95.3.7 show()

```cpp
def Fl_Overlay_Window::show () [virtual]
```

Makes a widget visible.

An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other events like FL_SHOW.

The `visible()` method returns true if the widget is set to be visible. The `visible_r()` method returns true if the widget and all of its parents are visible. A widget is only visible if `visible()` is true on it and all of its parents.

Changing it will send FL_SHOW or FL_HIDE events to the widget. Do not change it if the parent is not visible, as this will send false FL_SHOW or FL_HIDE events to the widget. `redraw()` is called if necessary on this or the parent.

See also

hide(), visible(), visible_r() 

Reimplemented from Fl_Double_Window.

The documentation for this class was generated from the following files:

- Fl_Overlay_Window.H
- Fl_Overlay_Window.cxx

12.96 Fl_Pack Class Reference

This widget was designed to add the functionality of compressing and aligning widgets.

```cpp
#include <Fl_Pack.H>
```

Inheritance diagram for Fl_Pack:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Pack
```

Public Types

- enum { VERTICAL = 0, HORIZONTAL = 1 }

Public Member Functions

- `Fl_Pack (int X, int Y, int W, int H, const char ∗L=0)`
  
  Creates a new Fl_Pack widget using the given position, size, and label string.

- `uchar horizontal () const`

  Returns non-zero if Fl_Pack alignment is horizontal.

- `void resize (int X, int Y, int W, int H)` FL_OVERRIDE

Generated by Doxygen
Override Fl_Group resize behavior.

- **int spacing()** const
  
  Gets the number of extra pixels of blank space that are added between the children.

- **void spacing(int i)**
  
  Sets the number of extra pixels of blank space that are added between the children.

### Public Member Functions inherited from Fl_Group

- **Fl_Widget *& _ddfdesign_kludge()**
  
  This is for forms compatibility only.

- **void add(Fl_Widget &)**
  
  The widget is removed from its current group (if any) and then added to the end of this group.

- **void add(Fl_Widget +o)**
  
  See void Fl_Group::add(Fl_Widget +w)

- **void add_resizable(Fl_Widget +o)**
  
  Adds a widget to the group and makes it the resizable widget.

- **Fl_Widget *const + array()** const
  
  Returns a pointer to the array of children.

- **Fl_Group const + as_group()** const FL_OVERRIDE
  
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- **void begin()**
  
  Sets the current group so you can build the widget tree by just constructing the widgets.

- **Fl_Widget + child(int n) const**
  
  Returns array()[n].

- **int children()** const
  
  Returns how many child widgets the group has.

- **void clear()**
  
  Deletes all child widgets from memory recursively.

- **unsigned int clip_children()**
  
  Returns the current clipping mode.

- **void clip_children(int c)**
  
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- **virtual int delete_child(int n)**
  
  Removes the widget at index from the group and deletes it.

- **void end()**
  
  Exactly the same as current(this->parent()).

- **int find(const Fl_Widget &o)** const
  
  See int Fl_Group::find(const Fl_Widget +w) const.

- **int find(const Fl_Widget +) const**
  
  Searches the child array for the widget and returns the index.

- **Fl_Group (int, int, int, int, const char *)=0**
  
  Creates a new Fl_Group widget using the given position, size, and label string.

- **void focus(Fl_Widget +W)**

- **void forms_end()**
  
  This is for forms compatibility only.

- **int handle(int) FL_OVERRIDE**
  
  Handles the specified event.

- **void init_sizes()**
  
  Resets the internal array of widget sizes and positions.

- **void insert(Fl_Widget &, int i)**
The widget is removed from its current group (if any) and then inserted into this group.

- void insert (Fl_Widget &, Fl_Widget *before)
  This does insert(w, find(before)).
- void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)
  Removes the widget o from the group.
- void remove (int index)
  Removes the widget at index from the group but does not delete it.
- Fl_Widget * resizable () const
  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)
  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()
  The destructor also deletes all the children.

### Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * as_gl_window () const
- virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * as_window () const
- void bind_deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
• void box (Fl_Boxtype new_box)
  
  Gets the box type of the widget.
• Fl_Callback_p callback () const
  
  Gets the current callback function for the widget.
• void callback (Fl_Callback *cb)
  
  Sets the current callback function for the widget.
• void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback *cb, void *p)
  
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 *cb)
  
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 *cb, long p=0)
  
  Sets the current callback function for the widget.
• unsigned int changed () const
  
  Checks if the widget value changed since the last callback.
• void clear_active ()
  
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  
  Clears or sets the damage flags.
• void clear_output ()
  
  Sets a widget to accept input.
• void clear_visible ()
  
  Hides the widget.
• void clear_visible_focus ()
  
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  
  Gets the background color of the widget.
• void color (Fl_Color bg)
  
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  
  For back compatibility only.
• void color2 (unsigned a)
  
  For back compatibility only.
• int contains (const Fl_Widget *w) const
  
  Checks if w is a child of this widget.
• void copy_label (const char *new_label)
  
  Sets the current label.
• void copy_tooltip (const char *text)
  
  Sets the current tooltip text.
• uchar damage () const
  
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  
  Sets the damage bits for an area inside the widget.
• **int damage_resize**(int, int, int, int)
  *Internal use only.*

• **void deactivate**()
  *Deactivates the widget.*

• **Fl_Image * deimage ()**
  *Gets the image that is used as part of the widget label when in the inactive state.*

• **const Fl_Image * deimage () const**
  *Gets the image that is used as part of the widget label when in the inactive state.*

• **void deimage(Fl_Image &img)**
  *Sets the image to use as part of the widget label when in the inactive state.*

• **void deimage(Fl_Image *img)**
  *Sets the image to use as part of the widget label when in the inactive state.*

• **int deimage_bound () const**
  *Returns whether the inactive image is managed by the widget.*

• **void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  *Calls the widget callback function with default arguments.*

• **void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  *Calls the widget callback function with arbitrary arguments.*

• **void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  *Calls the widget callback function with arbitrary arguments.*

• **void draw_label (int, int, int, int, Fl_Align) const**
  *Draws the label in an arbitrary bounding box with an arbitrary alignment.*

• **int h () const**
  *Gets the widget height.*

• **virtual void hide ()**
  *Makes a widget invisible.*

• **Fl_Image * image ()**
  *Gets the image that is used as part of the widget label when in the active state.*

• **const Fl_Image * image () const**
  *Gets the image that is used as part of the widget label when in the active state.*

• **void image (Fl_Image &img)**
  *Sets the image to use as part of the widget label when in the active state.*

• **void image (Fl_Image *img)**
  *Sets the image to use as part of the widget label when in the active state.*

• **int image_bound () const**
  *Returns whether the image is managed by the widget.*

• **int inside (const Fl_Widget *wgt) const**
  *Checks if this widget is a child of wgt.*

• **int is_label_copied () const**
  *Returns whether the current label was assigned with copy_label().*

• **const char * label () const**
  *Gets the current label text.*

• **void label (const char *text)**
  *Sets the current label pointer.*

• **void label (Fl_Labeltype a, const char *b)**
  *Shortcut to set the label text and type in one call.*

• **Fl_Color labelcolor () const**
  *Gets the label color.*

• **void labelcolor (Fl_Color c)**
  *Sets the label color.*

• **Fl_Font labelfont () const**
- Gets the font to use.
  - void labelfont (Fl_Font f)
    Sets the font to use.
- Fl_Fontsize labelsize () const
  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
- Fl_Labeltype labeltype () const
  Gets the label type.
- void labeltype (Fl_Labeltype a)
  Sets the label type.
- void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
- unsigned int output () const
  Returns if a widget is used for output only.
- Fl_Group * parent () const
  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)
  Internal use only - "for hacks only".
- void position (int X, int Y)
  Repositions the window or widget.
- void redraw ()
  Schedules the drawing of the widget.
- void redraw_label ()
  Schedules the drawing of the label.
- Fl_Color selection_color () const
  Gets the selection color.
- void selection_color (Fl_Color a)
  Sets the selection color.
- void set_active ()
  Marks the widget as active without sending events or changing focus.
- void set_changed ()
  Marks the value of the widget as changed.
- void set_output ()
  Sets a widget to output only.
- void set_visible ()
  Makes the widget visible.
- void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
- int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()
  Makes a widget visible.
- void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered 'x' shortcut.
• const char * tooltip () const
  Gets the current tooltip text.
• void tooltip (const char *text)
  Sets the current tooltip text.
• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.
Protected Member Functions

- void draw () FL_OVERRIDE
  Draws the widget.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.
- void draw_children ()
  Draws all children of the group.
- void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)
  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()
  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void draw_label () const
  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.
- unsigned int flags () const
  Gets the widget flags mask.
• void h (int v)  
  *Internal use only.*
• void set_flag (unsigned int c)  
  *Sets a flag in the flags mask.*
• void w (int v)  
  *Internal use only.*
• void x (int v)  
  *Internal use only.*
• void y (int v)  
  *Internal use only.*

**Additional Inherited Members**

**Static Public Member Functions inherited from Fl_Group**

• static Fl_Group ∗ current ()  
  *Returns the currently active group.*
• static void current (Fl_Group ∗ g)  
  *Sets the current group.*

**Static Public Member Functions inherited from Fl_Widget**

• static void default_callback (Fl_Widget ∗ widget, void ∗ data)  
  *The default callback for all widgets that don’t set a callback.*
• static unsigned int label_shortcut (const char ∗ t)  
  *Returns the Unicode value of the ‘&x’ shortcut in a given text.*
• static int test_shortcut (const char ∗ t, const bool require_alt=false)  
  *Returns true if the given text t contains the entered ‘&x’ shortcut.*

**Protected Types inherited from Fl_Widget**

• enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_relative = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}  
*flags possible values enumeration.*

### 12.96.1 Detailed Description

This widget was designed to add the functionality of compressing and aligning widgets. If `type()` is `Fl_Pack::HORIZONTAL` all the children are resized to the height of the Fl_Pack, and are moved next to each other horizontally. If `type()` is not `Fl_Pack::HORIZONTAL` then the children are resized to the width and are stacked below each other. Then the Fl_Pack resizes itself to surround the child widgets. You may want to put the Fl_Pack inside an Fl_Scroll.

The ‘resizable()’ for Fl_Pack is set to NULL by default. Its behavior is slightly different than in a normal Fl_Group widget: only if the resizable() widget is the last widget in the group it is extended to take the full available width or height, respectively, of the Fl_Pack group.
Note

You can nest Fl_Pack widgets or put them inside Fl_Scroll widgets or inside other group widgets but their behavior can sometimes be "surprising". This is partly due to the fact that Fl_Pack widgets resize themselves during their draw() operation, trying to react on their child widgets resizing themselves during their draw() operations which can be confusing. If you want to achieve special resize behavior of nested group widgets it can sometimes be easier to derive your own specialized group widget than to try to make nested Fl_Pack widgets behave as expected.

See also

Fl_Group::resizable()

12.96.2 Constructor & Destructor Documentation

12.96.2.1 Fl_Pack()

Fl_Pack::Fl_Pack (int X, int Y, int W, int H, const char * L = 0)

Creates a new Fl_Pack widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX.
The default type() is Fl_Pack::VERTICAL.
The destructor also deletes all the children. This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the Fl_Pack and all of its children can be automatic (local) variables, but you must declare the Fl_Pack first, so that it is destroyed last.

Parameters

- in X,Y X and Y coordinates (position)
- in W,H width and height, respectively
- in L label (optional)

12.96.3 Member Function Documentation

12.96.3.1 draw()

void Fl_Pack::draw ( ) [protected], [virtual]

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:
Fl_Widget * s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()

Reimplemented from Fl_Group.

12.96.3.2 horizontal()

uchar Fl_Pack::horizontal ( ) const [inline]

Returns non-zero if Fl_Pack alignment is horizontal.

Returns

- non-zero if Fl_Pack alignment is horizontal (Fl_Pack::HORIZONTAL)
Note

Currently the return value is the same as Fl_Group::type(), but this may change in the future. Do not set any other values than the following with Fl_Pack::type():

- Fl_Pack::VERTICAL (Default)
- Fl_Pack::HORIZONTAL

See class Fl_Pack documentation for details.

12.96.3.3 resize()

void Fl_Pack::resize (int X, int Y, int W, int H) [virtual]

Override Fl_Group resize behavior.
Resizing an Fl_Pack will not resize any of its children, but trigger a redraw, which in turn recalculates the dimensions of all children.

Parameters

| in   | X, Y, W, H | new position and size of the Fl_Pack widget |

Reimplemented from Fl_Group.
The documentation for this class was generated from the following files:

- Fl_Pack.H
- Fl_Pack.cxx

12.97 Fl_Paged_Device Class Reference

Represents page-structured drawing surfaces.
#include <Fl_Paged_Device.H>

Inheritance diagram for Fl_Paged_Device:

```
Fl_Paged_Device
    ▼
     |  Fl_Surface_Device
          ▼
           |  Fl_Widget_Surface
                    ▼
                     |  Fl_Paged_Device
                              ▼
                               Fl_PostScript_File_Device  Fl_Printer
```

Classes

- struct page_format
  width, height and name of a page format

Public Types

- enum Page_Format {
  A0 = 0, A1, A2, A3,
  A4, A5, A6, A7,
**A8, A9, B0, B1, B2, B3, B4, B5, B6, B7, B8, B9, B10, CSE, DLE, EXECUTIVE, FOLIO, LEDGER, LEGAL, LETTER, TABLOID, ENVELOPE, MEDIA = 0x1000**

Possible page formats.

- **enum Page_Layout { PORTRAIT = 0, LANDSCAPE = 0x100, REVERSED = 0x200, ORIENTATION = 0x300 }**

Possible page layouts.

**Public Member Functions**

- **virtual int begin_job (int pagecount=0, int *frompage=NULL, int *topage=NULL, char **perr_message=NULL)**
  - Begins a print job.

- **virtual int begin_page (void)**
  - Begins a new printed page.

- **virtual void end_job (void)**
  - To be called at the end of a print job.

- **virtual int end_page (void)**
  - To be called at the end of each page.

- **virtual void margins (int *left, int *top, int *right, int *bottom)**
  - Computes the dimensions of margins that lie between the printable page area and the full page.

- **void print_widget (Fl_Widget *widget, int delta_x=0, int delta_y=0)**
  - Synonym of draw(Fl_Widget*, int, int)

- **void print_window (Fl_Window *win, int x_offset=0, int y_offset=0)**
  - Synonym of draw_decorated_window(Fl_Window*, int, int)

- **virtual void rotate (float angle)**
  - Rotates the graphics operations relatively to paper.

- **virtual void scale (float scale_x, float scale_y=0.)**
  - Changes the scaling of page coordinates.

- **int start_job (int pagecount=0, int *frompage=NULL, int *topage=NULL, char **perr_message=NULL)**
  - Synonym of begin_job(int pagecount, int *frompage, int *topage, char **perr_message).

- **int start_page ()**
  - Synonym of begin_page().

- **virtual ~Fl_Paged_Device ()**
  - The destructor.

**Public Member Functions inherited from Fl_Widget_Surface**

- **void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)**
  - Draws the widget on the drawing surface.

- **void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)**
  - Draws a window with its title bar and frame if any.

- **virtual void origin (int *x, int *y)**
  - Computes the coordinates of the current origin of graphics functions.

- **virtual void origin (int x, int y)**
  - Sets the position of the origin of graphics in the drawable part of the drawing surface.

- **void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)**
  - Draws a rectangular part of an on-screen window.

- **virtual int printable_rect (int *w, int *h)**
Computes the width and height of the drawable area of the drawing surface.

- virtual void translate (int x, int y)
  Translates the current graphics origin accounting for the current rotation.
- virtual void untranslate ()
  Undoes the effect of a previous translate() call.

Public Member Functions inherited from Fl_Surface_Device

- Fl/Graphics_Driver * driver ()
  Returns the graphics driver of this drawing surface.
- virtual bool is_current ()
  Is this surface the current drawing surface?
- virtual void set_current (void)
  Make this surface the current drawing surface.
- virtual ~Fl_Surface_Device ()
  The destructor.

Static Public Attributes

- static const page_format page_formats [NO_PAGE_FORMATS]
  width, height and name of all elements of the enum Page_Format.

Protected Member Functions

- Fl_Paged_Device ()
  The constructor.

Protected Member Functions inherited from Fl_Widget_Surface

- Fl_Widget_Surface (Fl_Graphics_Driver * d)
  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- void driver (Fl_Graphics_Driver * graphics_driver)
  Sets the graphics driver of this drawing surface.
- virtual void end_current ()
  FLTK calls this each time a surface ceases to be the current drawing surface.
- Fl_Surface_Device (Fl_Graphics_Driver * graphics_driver)
  Constructor that sets the graphics driver to use for the created surface.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()
  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device * new_current)
  Pushes new_current on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()
  The current drawing surface.
Protected Attributes inherited from Fl_Widget_Surface

- int x_offset
  
  horizontal offset to the origin of graphics coordinates

- int y_offset
  
  vertical offset to the origin of graphics coordinates

12.97.1 Detailed Description

Represents page-structured drawing surfaces.
This class has no public constructor: don't instantiate it; use Fl_Printer or Fl_PostScript_File_Device instead.

12.97.2 Member Enumeration Documentation

12.97.2.1 Page_Format

defined in Fl_Paged_Device.h

Possible page formats.
All paper formats with pre-defined width and height. The Fl_Paged_Device::page_formats array gives these widths and heights.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>A0 format</td>
</tr>
<tr>
<td>A1</td>
<td>A1 format</td>
</tr>
<tr>
<td>A2</td>
<td>A2 format</td>
</tr>
<tr>
<td>A3</td>
<td>A3 format</td>
</tr>
<tr>
<td>A4</td>
<td>A4 format</td>
</tr>
<tr>
<td>A5</td>
<td>A5 format</td>
</tr>
<tr>
<td>A6</td>
<td>A6 format</td>
</tr>
<tr>
<td>A7</td>
<td>A7 format</td>
</tr>
<tr>
<td>A8</td>
<td>A8 format</td>
</tr>
<tr>
<td>A9</td>
<td>A9 format</td>
</tr>
<tr>
<td>B0</td>
<td>B0 format</td>
</tr>
<tr>
<td>B1</td>
<td>B1 format</td>
</tr>
<tr>
<td>B2</td>
<td>B2 format</td>
</tr>
<tr>
<td>B3</td>
<td>B3 format</td>
</tr>
<tr>
<td>B4</td>
<td>B4 format</td>
</tr>
<tr>
<td>B5</td>
<td>B5 format</td>
</tr>
<tr>
<td>B6</td>
<td>B6 format</td>
</tr>
<tr>
<td>B7</td>
<td>B7 format</td>
</tr>
<tr>
<td>B8</td>
<td>B8 format</td>
</tr>
<tr>
<td>B9</td>
<td>B9 format</td>
</tr>
<tr>
<td>B10</td>
<td>B10 format</td>
</tr>
<tr>
<td>EXECUTIVE</td>
<td>Executive format</td>
</tr>
<tr>
<td>FOLIO</td>
<td>Folio format</td>
</tr>
<tr>
<td>LEDGER</td>
<td>Ledger format</td>
</tr>
<tr>
<td>LEGAL</td>
<td>Legal format</td>
</tr>
<tr>
<td>LETTER</td>
<td>Letter format</td>
</tr>
<tr>
<td>TABLOID</td>
<td>Tabloid format</td>
</tr>
</tbody>
</table>

12.97.2.2 Page_Layout

defined in Fl_Paged_Device.h

Generated by Doxygen
Possible page layouts.

Enumerator

| PORTRAIT | Portrait orientation. |
| LANDSCAPE | Landscape orientation. |
| REVERSED | Reversed orientation. |
| ORIENTATION | orientation |

12.97.3 Member Function Documentation

12.97.3.1 begin_job()

```cpp
int Fl_Paged_Device::begin_job (  
    int pagecount = 0,
    int * frompage = NULL,
    int * topage = NULL,
    char ** perr_message = NULL ) [virtual]
```

Begins a print job.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>pagecount</th>
<th>the total number of pages of the job (or 0 if you don’t know the number of pages)</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>frompage</td>
<td>if non-null, *frompage is set to the first page the user wants printed</td>
</tr>
<tr>
<td>out</td>
<td>topage</td>
<td>if non-null, *topage is set to the last page the user wants printed</td>
</tr>
<tr>
<td>out</td>
<td>perr_message</td>
<td>if non-null and if the returned value is 2, *perr_message is set to a string describing the error. That string can be delete[]’d after use.</td>
</tr>
</tbody>
</table>

Returns

0 if OK, 1 if user cancelled the job, 2 if any error.

Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.

12.97.3.2 begin_page()

```cpp
int Fl_Paged_Device::begin_page (  
    void ) [virtual]
```

Begins a new printed page.

The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area. This function also makes this surface the current drawing surface with Fl_Surface_Device::push_current().

Note

begin_page() calls Fl_Surface_Device::push_current() and leaves this device as the active surface. If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current(), or the content of the dialog box will be rendered to the printer instead of the screen.

Returns

0 if OK, non-zero if any error

Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.
12.97.3.3  
end_job()

```cpp
void Fl_Paged_Device::end_job (  
    void ) [virtual]
```

To be called at the end of a print job.
Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.

12.97.3.4  
end_page()

```cpp
int Fl_Paged_Device::end_page (  
    void ) [virtual]
```

To be called at the end of each page.
This function also stops this surface from being the current drawing surface with Fl_Surface_Device::pop_current().

**Note**

end_page() calls Fl_Surface_Device::pop_current(). If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current().

**Returns**

0 if OK, non-zero if any error.

Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.

12.97.3.5  margins()

```cpp
void Fl_Paged_Device::margins (  
    int * left,  
    int * top,  
    int * right,  
    int * bottom ) [virtual]
```

Computes the dimensions of margins that lie between the printable page area and the full page.
Values are in the same unit as that used by FLTK drawing functions. They are changed by scale() calls.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out left</code></td>
<td>If non-null, <code>*left</code> is set to the left margin size.</td>
</tr>
<tr>
<td><code>out top</code></td>
<td>If non-null, <code>*top</code> is set to the top margin size.</td>
</tr>
<tr>
<td><code>out right</code></td>
<td>If non-null, <code>*right</code> is set to the right margin size.</td>
</tr>
<tr>
<td><code>out bottom</code></td>
<td>If non-null, <code>*bottom</code> is set to the bottom margin size.</td>
</tr>
</tbody>
</table>

Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.

12.97.3.6  rotate()

```cpp
void Fl_Paged_Device::rotate (  
    float angle ) [virtual]
```

Rotates the graphics operations relatively to paper.
The rotation is centered on the current graphics origin. Successive rotate() calls don’t combine their effects.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>angle</code></td>
<td>Rotation angle in counter-clockwise degrees.</td>
</tr>
</tbody>
</table>

Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.
12.97.3.7 scale()

void Fl_Paged_Device::scale ( 
    float scale_x, 
    float scale_y = 0. ) [virtual]

Changes the scaling of page coordinates.
This function also resets the origin of graphics functions at top left of printable page area. After a scale() call, do a printable_rect() call to get the new dimensions of the printable page area. Successive scale() calls don't combine their effects.

Parameters

| scale←-_x | Horizontal dimensions of plot are multiplied by this quantity. |
| scale←-_y | Same as above, vertically. The value 0. is equivalent to setting scale_y = scale_x. Thus, scale(factor); is equivalent to scale(factor, factor); |

Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.

12.97.3.8 start_job()

int Fl_Paged_Device::start_job ( 
    int pagecount = 0, 
    int * frompage = NULL, 
    int * topage = NULL, 
    char ** perr_message = NULL ) [inline]

Synonym of begin_job(int pagecount, int *frompage, int *topage, char **perr_message).
For API compatibility with FLTK 1.3.x

12.97.3.9 start_page()

int Fl_Paged_Device::start_page () [inline]
Synonym of begin_page().
For API compatibility with FLTK 1.3.x

The documentation for this class was generated from the following files:

- Fl_Paged_Device.H
- Fl_Paged_Device.cxx

12.98 Fl_Pixmap Class Reference

The Fl_Pixmap class supports caching and drawing of colormap ( pixmap) images, including transparency.
#include <Fl_Pixmap.H>

Inheritance diagram for Fl_Pixmap:

```
  Fl_Image
   |
   v
  Fl_Pixmap
     |
     v
  Fl_GIF_Image Fl_XPM_Image
     |
     v
Fl_Anim_GIF_Image
```
Public Member Functions

- int cache_h ()
- int cache_w ()
- void color_average (FL_Color c, float i) FL_OVERRIDE
  
  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * copy () const
- Fl_Image * copy (int W, int H) const FL_OVERRIDE

  Creates a resized copy of the image.
- void desaturate () FL_OVERRIDE

  The desaturate() method converts an image to grayscale.
- void draw (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE
  
  Draws the image to the current drawing surface with a bounding box.
- Fl_Pixmap (char *const *D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (const char *const *D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (const uchar *const *D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (uchar *const *D)

  The constructors create a new pixmap from the specified XPM data.
- void label (Fl_Menu_Item *m) FL_OVERRIDE

  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget *w) FL_OVERRIDE

  This method is an obsolete way to set the image attribute of a widget or menu item.
- void uncache () FL_OVERRIDE

  If the image has been cached for display, delete the cache data.
- virtual ~Fl_Pixmap ()

  The destructor frees all memory and server resources that are used by the pixmap.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image * as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image * copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char *const * data () const

  Returns a pointer to the current image data array.
- int data_h () const

  Returns the height of the image data.
- int data_w () const

  Returns the width of the image data.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)
The constructor creates an empty image with the specified width, height, and depth.

- `int h () const`
  Returns the current image drawing height in FLTK units.

- `void inactive ()`
  The inactive() method calls `color_average(FL_BACKGROUND_COLOR, 0.33)` to produce an image that appears grayed out.

- `int ld () const`
  Returns the current line data size in bytes.

- `virtual void release ()`
  Releases an `Fl_Image` - the same as 'delete this'.

- `virtual void scale (int width, int height, int proportional=1, int can_expand=0)`
  Sets the drawing size of the image.

- `int w () const`
  Returns the current image drawing width in FLTK units.

- `virtual ~Fl_Image ()`
  The destructor is a virtual method that frees all memory used by the image.

Public Attributes

- `int alloc_data`

Protected Member Functions

- `void measure ()`

Protected Member Functions inherited from `Fl_Image`

- `void d (int D)`
  Sets the current image depth.

- `void data (const char *const *p, int c)`
  Sets the current data pointer and count of pointers in the array.

- `void draw_empty (int X, int Y)`
  The protected method draw_empty() draws a box with an X in it.

- `int draw_scaled (int X, int Y, int W, int H)`
  Draw the image to the current drawing surface rescaled to a given width and height.

- `void h (int H)`
  Sets the height of the image data.

- `void ld (int LD)`
  Sets the current line data size in bytes.

- `void w (int W)`
  Sets the width of the image data.

Friends

- `class Fl_Graphics_Driver`

Additional Inherited Members

Static Public Member Functions inherited from `Fl_Image`

- `static Fl_Labeltype define_FL_IMAGE_LABEL ()`

- `static Fl_RGB_Scaling RGB_scaling ()`
  Returns the currently used RGB image scaling method.

- `static void RGB_scaling (Fl_RGB_Scaling)`

Generated by Doxygen
Sets the RGB image scaling method used for copy(int, int).

- static Fl_RGB_Scaling scaling_algorithm ()

    Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

    Sets what algorithm is used when resizing a source image to draw it.

Static Public Attributes inherited from Fl_Image

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_MEMORY_ACCESS = -4
- static const int ERR_NO_IMAGE = -1
- static bool register_images_done = false

    True after fl_register_images() was called, false before.

Static Protected Member Functions inherited from Fl_Image

- static void labeltype (const Fl_Label ∗lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void measure (const Fl_Label ∗lo, int &lw, int &lh)

12.98.1 Detailed Description

The Fl_Pixmap class supports caching and drawing of colormap ( pixmap) images, including transparency.

12.98.2 Constructor & Destructor Documentation

12.98.2.1 Fl_Pixmap() [1/4]

Fl_Pixmap::Fl_Pixmap ( char ∗const ∗D ) [inline], [explicit]

The constructors create a new pixmap from the specified XPM data.

12.98.2.2 Fl_Pixmap() [2/4]

Fl_Pixmap::Fl_Pixmap ( uchar ∗const ∗D ) [inline], [explicit]

The constructors create a new pixmap from the specified XPM data.

12.98.2.3 Fl_Pixmap() [3/4]

Fl_Pixmap::Fl_Pixmap ( const char ∗const ∗D ) [inline], [explicit]

The constructors create a new pixmap from the specified XPM data.

12.98.2.4 Fl_Pixmap() [4/4]

Fl_Pixmap::Fl_Pixmap ( const uchar ∗const ∗D ) [inline], [explicit]

The constructors create a new pixmap from the specified XPM data.
12.98.3 Member Function Documentation

12.98.3.1 color_average()

```cpp
void Fl_Pixmap::color_average ( Fl_Color c, float i ) [virtual]
```

The `color_average()` method averages the colors in the image with the provided FLTK color value. The first argument specifies the FLTK color to be used. The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color. An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory. Reimplemented from `Fl_Image`.

12.98.3.2 copy()

```cpp
Fl_Image * Fl_Pixmap::copy ( int W, int H ) const [virtual]
```

Creates a resized copy of the image. The new image should be released when you are done with it. Note: since FLTK 1.4.0 you can use `Fl_Image::release()` for all types of images (i.e. all subclasses of `Fl_Image`) instead of `operator delete` for `Fl_Image`s and `Fl_Image::release()` for `Fl_Shared_Image`'s. The new image data will be converted to the requested size. RGB images are resized using the algorithm set by `Fl_Image::RGB_scaling()`.

For the new image the following equations are true:

- \( w() = \text{data}_w() = W \)
- \( h() = \text{data}_h() = H \)

**Parameters**

| in | W/H | Requested width and height of the new image |

**Note**

The returned image can be safely cast to the same image type as that of the source image provided this type is one of `Fl_RGB_Image`, `Fl_SVG_Image`, `Fl_Pixmap`, `Fl_Bitmap`, `Fl_Tiled_Image`, `Fl_Anim_GIF_Image` and `Fl_Shared_Image`. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an `Fl_RGB_Image` is an object of class `Fl_Pixmap`.

Since FLTK 1.4.0 this method is `const`. If you derive your own class from `Fl_Image` or any subclass your overridden methods of `Fl_Image::copy() const` and `Fl_Image::copy(int, int) const` **must** also be `const` for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been `const`.

Reimplemented from `Fl_Image`.

12.98.3.3 desaturate()

```cpp
void Fl_Pixmap::desaturate ( ) [virtual]
```

The `desaturate()` method converts an image to grayscale. If the image contains an alpha channel (depth = 4), the alpha channel is preserved. An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory. Reimplemented from `Fl_Image`.
12.98.3.4 draw()

```cpp
void Fl_Pixmap::draw (  
    int X,  
    int Y,  
    int W,  
    int H,  
    int cx = 0,  
    int cy = 0 ) [virtual]
```

Draws the image to the current drawing surface with a bounding box. Arguments X, Y, W, H specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the cx and cy arguments. In other words: `fl_push_clip(X,Y,W,H)` is applied, the image is drawn with its upper-left corner at X-cx,Y-cy and its own width and height, `fl_pop_clip()` is applied. Reimplemented from Fl_Image.

12.98.3.5 label() [1/2]

```cpp
void Fl_Pixmap::label (  
    Fl_Menu_Item * m ) [virtual]
```

This method is an obsolete way to set the image attribute of a menu item. **Deprecated** Please use `Fl_Menu_Item::image()` instead.

Reimplemented from Fl_Image.

12.98.3.6 label() [2/2]

```cpp
void Fl_Pixmap::label (  
    Fl_Widget * widget ) [virtual]
```

This method is an obsolete way to set the image attribute of a widget or menu item. **Deprecated** Please use `Fl_Widget::image()` or `Fl_Widget::deimage()` instead.

Reimplemented from Fl_Image.

12.98.3.7 uncache()

```cpp
void Fl_Pixmap::uncache ( ) [virtual]
```

If the image has been cached for display, delete the cache data. This allows you to change the data used for the image and then redraw it without recreating an image object. Reimplemented from Fl_Image.

The documentation for this class was generated from the following files:

- Fl_Pixmap.H
- Fl_Pixmap.cxx

12.99 Fl_Plugin Class Reference

**Fl_Plugin** allows link-time and run-time integration of binary modules.

```cpp
#include <Fl_Plugin.H>
```

Inheritance diagram for Fl_Plugin:

```
Fl_Plugin
```

```
Fl_Device_Plugin
```

Generated by Doxygen
Public Member Functions

- Fl_Plugin (const char *klass, const char *name)
  Create a plugin.
- virtual ~Fl_Plugin ()
  Clear the plugin and remove it from the database.

12.99.1 Detailed Description

Fl_Plugin allows link-time and run-time integration of binary modules.
Fl_Plugin and Fl_Plugin_Manager provide a small and simple solution for linking C++ classes at run-time, or optionally linking modules at compile time without the need to change the main application.
Fl_Plugin_Manager uses static initialization to create the plugin interface early during startup. Plugins are stored in a temporary database, organized in classes.
Plugins should derive a new class from Fl_Plugin as a base:

```cpp
class My_Plugin : public Fl_Plugin {
public:
  My_Plugin() : Fl_Plugin("effects", "blur") { }
  void do_something(...);
};
My_Plugin blur_plugin();
```

Plugins can be put into modules and either linked before distribution, or loaded from dynamically linkable files. An Fl_Plugin_Manager is used to list and access all currently loaded plugins.

```cpp
Fl_Plugin_Manager mgr("effects");
int i, n = mgr.plugins();
for (i=0; i<n; i++) {
  My_Plugin *pin = (My_Plugin*)mgr.plugin(i);
  pin->do_something();
}
```

12.99.2 Constructor & Destructor Documentation

12.99.2.1 Fl_Plugin()

Fl_Plugin::Fl_Plugin (const char * klass, const char * name )

Create a plugin.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>klass</th>
<th>plugins are grouped in classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>every plugin should have a unique name</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Plugin.H
- Fl_Preferences.cxx

12.100 Fl_Plugin_Manager Class Reference

Fl_Plugin_Manager manages link-time and run-time plugin binaries.

```cpp
#include "Fl_Plugin.H"
```
Inheritance diagram for Fl_Plugin_Manager:

```
FlPreferences

Fl_Plugin_Manager
```
Public Member Functions

- **Fl_Preferences::ID addPlugin (const char *name, Fl_Plugin *plugin)**
  
  This function adds a new plugin to the database.

- **Fl_Plugin_Manager (const char *klass)**
  
  Manage all plugins belonging to one class.

- **Fl_Plugin * plugin (const char *name)**
  
  Return the address of a plugin by name.

- **Fl_Plugin * plugin (int index)**
  
  Return the address of a plugin by index.

- **int plugins ()**
  
  Return the number of plugins in the klass.

- **~Fl_Plugin_Manager ()**
  
  Remove the plugin manager.

Public Member Functions inherited from Fl_Preferences

- **char clear ()**
  
  Delete all groups and all entries.

- **char delete_all_entries ()**
  
  Delete all entries.

- **char delete_all_groups ()**
  
  Delete all groups.

- **char delete_entry (const char *entry)**
  
  Deletes a single name/value pair.

- **char delete_group (const char *group)**
  
  Deletes a group.

- **int dirty ()**
  
  Check if there were changes to the database that need to be written to disk.

- **int entries ()**
  
  Returns the number of entries (name/value pairs) in a group.

- **const char * entry (int index)**
  
  Returns the name of an entry.

- **char entry_exists (const char *key)**
  
  Returns non-zero if an entry with this name exists.

- **Root filename (char *buffer, size_t buffer_size)**
  
  Return the file name and path to the preference file.

- **Fl_Preferences (const char *path, const char *vendor, const char *application)**
  
  Deprecated: Use this constructor to create or read a preference file at an arbitrary position in the file system.

- **Fl_Preferences (const char *path, const char *vendor, const char *application, Root flags)**
  
  Use this constructor to create or read a preference file at an arbitrary position in the file system.

- **Fl_Preferences (const Fl_Preferences &)**
  
  Create another reference to a Preferences group.

- **Fl_Preferences (Fl_Preferences &parent, const char *group)**
  
  Generate or read a new group of entries within another group.

- **Fl_Preferences (Fl_Preferences &parent, int groupIndex)**
  
  Open a child group using a given index.

- **Fl_Preferences (Fl_Preferences *parent, const char *group)**
  
  Create or access a group of preferences using a name.

- **Fl_Preferences (Fl_Preferences *parent, int groupIndex)**

- **Fl_Preferences (ID id)**
  
  Create a new dataset access point using a dataset ID.
• **FL_Preferences (Root root, const char *vendor, const char *application)**

  The constructor creates a group that manages key/value pairs and child groups.

• **int flush ()**

  Writes preferences to disk if they were modified.

• **char get (const char *entry, char *value, const char *defaultValue)**

  Reads an entry from the group.

• **char get (const char *entry, char *value, const char *defaultValue, int maxSize)**

  Reads an entry from the group.

• **char get (const char *entry, double &value, double defaultValue)**

  Reads an entry from the group.

• **char get (const char *entry, float &value, float defaultValue)**

  Reads an entry from the group.

• **char get (const char *entry, int &value, int defaultValue)**

  Reads an entry from the group.

• **char get (const char *entry, void *value, const void *defaultValue, int defaultSize)**

  Reads an entry from the group.

• **char get (const char *entry, void *value, const void *defaultValue, int defaultSize, int *size)**

  Reads a binary entry from the group, encoded in hexadecimal blocks.

• **char get_userdata_path (char *path, int pathlen)**

  Creates a path that is related to the preference file and that is usable for additional application data.

• **const char * group (int num_group)**

  Returns the name of the Nth (num_group) group.

• **char group_exists (const char *key)**

  Returns non-zero if a group with this name exists.

• **int groups ()**

  Returns the number of groups that are contained within a group.

• **ID id ()**

  Return an ID that can later be reused to open more references to this dataset.

• **const char * name ()**

  Return the name of this entry.

• **const char * path ()**

  Return the full path to this entry.

• **char set (const char *entry, const char *value)**

  Sets an entry (name/value pair).

• **char set (const char *entry, const void *value, int size)**

  Sets an entry (name/value pair).

• **char set (const char *entry, double value)**

  Sets an entry (name/value pair).

• **char set (const char *entry, double value, int precision)**

  Sets an entry (name/value pair).

• **char set (const char *entry, float value)**

  Sets an entry (name/value pair).

• **char set (const char *entry, float value, int precision)**

  Sets an entry (name/value pair).

• **char set (const char *entry, int value)**

  Sets an entry (name/value pair).

• **int size (const char *entry)**

  Returns the size of the value part of an entry.

• **virtual ~FL_Preferences ()**

  The destructor removes allocated resources.
Static Public Member Functions

- static int load (const char *filename)
  Load a module from disk.
- static int loadAll (const char *dirpath, const char *pattern=0)
  Use this function to load a whole directory full of modules.
- static void removePlugin (Fl_Preferences::ID id)
  Remove any plugin.

Static Public Member Functions inherited from Fl_Preferences

- static unsigned int file_access ()
  Return the current file access permissions for the FLTK preferences system.
- static void file_access (unsigned int flags)
  Tell the FLTK preferences system which files in the file system it may read, create, or write.
- static Root filename (char *buffer, size_t buffer_size, Root root, const char *vendor, const char *application)
  Determine the file name and path to preferences that would be opened with these parameters.
- static const char *new_UUID ()
  Returns a UUID as generated by the system.
- static char remove (ID id_)
  Remove the group with this ID from a database.

Additional Inherited Members

Public Types inherited from Fl_Preferences

- typedef void *ID
  Every Fl_Preferences-Group has a unique ID.
- enum Root {
  UNKNOWN_ROOT_TYPE = -1 , SYSTEM = 0 , USER , MEMORY ,
  ROOT_MASK = 0x00FF , CORE = 0x0100 , C_LOCALE = 0x1000 , CLEAR = 0x2000 ,
  SYSTEM_L = SYSTEM | C_LOCALE , USER_L = USER | C_LOCALE , CORE_SYSTEM_L = CORE |
  SYSTEM_L , CORE_USER_L = CORE | USER ,
  CORE_SYSTEM = CORE | SYSTEM , CORE_USER = CORE | USER }
  Define the scope of the preferences.

Static Public Attributes inherited from Fl_Preferences

- static const unsigned int ALL = ALL_READ_OK | ALL_WRITE_OK
  Set this to give FLTK and applications permission to read, write, and create preference files.
- static const unsigned int ALL_READ_OK = USER_READ_OK | SYSTEM_READ_OK | CORE_READ_OK
  Set this to allow FLTK and applications to read preference files.
- static const unsigned int ALL_WRITE_OK = USER_WRITE_OK | SYSTEM_WRITE_OK | CORE_WRITE_OK
  Set this to allow FLTK and applications to create and write preference files.
- static const unsigned int APP_OK = SYSTEM_OK | USER_OK
  Set this if it is OK for applications to read, create, and write any kind of preference files.
- static const unsigned int CORE_OK = CORE_READ_OK | CORE_WRITE_OK
  Set this if it is OK for FLTK to read, create, or write preference files.
- static const unsigned int CORE_READ_OK = 0x0010
  Set this if it is OK for FLTK to read preference files.
- static const unsigned int CORE_WRITE_OK = 0x0020
  Set this if it is OK for FLTK to create or write preference files.
- static const unsigned int NONE = 0x0000
  Set this if no call to Fl_Preferences shall access the file system.
• static const unsigned int SYSTEM_OK = SYSTEM_READ_OK | SYSTEM_WRITE_OK
  Set this if it is OK for applications to read, create, and write system wide preference files.
• static const unsigned int SYSTEM_READ_OK = 0x0004
  Set this if it is OK for applications to read system wide preference files.
• static const unsigned int SYSTEM_WRITE_OK = 0x0008
  Set this if it is OK for applications to create and write system wide preference files.
• static const unsigned int USER_OK = USER_READ_OK | USER_WRITE_OK
  Set this if it is OK for applications to read, create, and write user preference files.
• static const unsigned int USER_READ_OK = 0x0001
  Set this if it is OK for applications to read user preference files.
• static const unsigned int USER_WRITE_OK = 0x0002
  Set this if it is OK for applications to create and write user preference files.

Protected Attributes inherited from Fl_Preferences

• Node * node
• RootNode * rootNode

12.100.1 Detailed Description

Fl_Plugin_Manager manages link-time and run-time plugin binaries.

See also

Fl_Plugin

12.100.2 Constructor & Destructor Documentation

12.100.2.1 ~Fl_Plugin_Manager()

Fl_Plugin_Manager::~Fl_Plugin_Manager ( )
Remove the plugin manager.
Calling this does not remove the database itself or any plugins. It just removes the reference to the database.

12.100.3 Member Function Documentation

12.100.3.1 addPlugin()

Fl_Preferences::ID Fl_Plugin_Manager::addPlugin ( const char * name,
             Fl_Plugin * plugin )
This function adds a new plugin to the database.
There is no need to call this function explicitly. Every Fl_Plugin constructor will call this function at initialization time.

12.100.3.2 load()

int Fl_Plugin_Manager::load ( const char * filename ) [static]
Load a module from disk.
A module must be a dynamically linkable file for the given operating system. When loading a module, its init function will be called which in turn calls the constructor of all statically initialized Fl_Plugin classes and adds them to the database.
12.100.3.3 loadAll()

```cpp
int Fl_Plugin_Manager::loadAll (  
    const char * dirpath,
    const char * pattern = 0 ) [static]
```

Use this function to load a whole directory full of modules.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dirpath</strong></td>
<td>Pathname of a directory. It <strong>must</strong> end with the platform's directory separator character (i.e., '\' under Windows, '/' otherwise).</td>
</tr>
<tr>
<td><strong>pattern</strong></td>
<td>A filename pattern to catch all modules of interest in the targeted directory (e.g., &quot;*.so,*.dll,*.dylib&quot;), or NULL to catch all files in the directory.</td>
</tr>
</tbody>
</table>

## 12.100.3.4 removePlugin()

```cpp
def Fl_Plugin_Manager::removePlugin (Fl_Preferences::ID id) [static]
```

Remove any plugin.

There is no need to call this function explicitly. Every Fl_Plugin destructor will call this function at destruction time.

The documentation for this class was generated from the following files:
- Fl_Plugin.H
- Fl_Preferences.cxx

## 12.101 Fl_PNG_Image Class Reference

The Fl_PNG_Image class supports loading, caching, and drawing of Portable Network Graphics (PNG) image files.

```cpp
#include <Fl_PNG_Image.H>
```

Inheritance diagram for Fl_PNG_Image:

```
Fl_PNG_Image
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_RGB_Image</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>Fl_Image</td>
</tr>
</tbody>
</table>
```

### Public Member Functions

- **Fl_PNG_Image (const char *filename)**
  
  *The constructor loads the named PNG image from the given png filename.*

- **Fl_PNG_Image (const char *name_png, const unsigned char *buffer, int datasize)**

  *Constructor that reads a PNG image from memory.*

### Public Member Functions inherited from Fl_RGB_Image

- virtual **Fl_SVG_Image * as_svg_image ()**
  
  *Returns whether an image is an Fl_SVG_Image or not.*

- int **cache_h ()**

- int **cache_w ()**

- void **color_average (Fl_Color c, float i) FL_OVERRIDE**
  
  *The color_average() method averages the colors in the image with the provided FLTK color value.*

- **Fl_Image * copy () const**

- **Fl_Image * copy (int W, int H) const FL_OVERRIDE**
  
  *Creates a resized copy of the image.*

- void **desaturate () FL_OVERRIDE**
  
  *The desaturate() method converts an image to grayscale.*

- void **draw (int X, int Y)**
• void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE
  Draws the image to the current drawing surface with a bounding box.

• Fl_RGB_Image (const Fl_Pixmap *pxm, Fl_Color bg=FL_GRAY)
  The constructor creates a new RGBA image from the specified Fl_Pixmap.

• Fl_RGB_Image (const uchar *bits, int bits_length, int W, int H, int D, int LD)
  The constructor creates a new image from the specified data.

• Fl_RGB_Image (const uchar *bits, int W, int H, int D=3, int LD=0)
  The constructor creates a new image from the specified data.

• void label (Fl_Menu_Item *m) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a menu item.

• void label (Fl_Widget *w) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a widget or menu item.

• virtual void normalize ()
  Makes sure the object is fully initialized.

• void uncache () FL_OVERRIDE
  If the image has been cached for display, delete the cache data.

• ~Fl_RGB_Image () FL_OVERRIDE
  The destructor frees all memory and server resources that are used by the image.

Public Member Functions inherited from Fl_Image

• virtual class Fl_Shared_Image * as_shared_image ()
  Returns whether an image is an Fl_Shared_Image or not.

• Fl_Image * copy () const
  Creates a copy of the image in the same size.

• int count () const
  Returns the number of data values associated with the image.

• int d () const
  Returns the image depth.

• const char *const * data () const
  Returns a pointer to the current image data array.

• int data_h () const
  Returns the height of the image data.

• int data_w () const
  Returns the width of the image data.

• void draw (int X, int Y)
  Draws the image to the current drawing surface.

• int fail () const
  Returns a value that is not 0 if there is currently no image available.

• Fl_Image (int W, int H, int D)
  The constructor creates an empty image with the specified width, height, and depth.

• int h () const
  Returns the current image drawing height in FLTK units.

• void inactive ()
  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

• int id () const
  Returns the current line data size in bytes.

• virtual void release ()
  Releases an Fl_Image - the same as 'delete this'.

• virtual void scale (int width, int height, int proportional=1, int can_expand=0)
Sets the drawing size of the image.

- `int w () const`
  Returns the current image drawing width in FLTK units.

- `virtual ~Fl_Image ()`
  The destructor is a virtual method that frees all memory used by the image.

Friends

- class `Fl_ICO_Image`

Additional Inherited Members

Static Public Member Functions inherited from `Fl_RGB_Image`

- `static size_t max_size ()`
  Returns the maximum allowed image size in bytes when creating an `Fl_RGB_Image` object.

- `static void max_size (size_t size)`
  Sets the maximum allowed image size in bytes when creating an `Fl_RGB_Image` object.

Static Public Member Functions inherited from `Fl_Image`

- `static Fl_Labeltype define_FL_IMAGE_LABEL ()`
  Returns the currently used RGB image scaling method.

- `static Fl_RGB_Scaling RGB_scaling ()`
  Sets the RGB image scaling method used for `copy(int, int)`.

- `static Fl_RGB_Scaling scaling_algorithm ()`
  Gets what algorithm is used when resizing a source image to draw it.

- `static void scaling_algorithm (Fl_RGB_Scaling algorithm)`
  Sets what algorithm is used when resizing a source image to draw it.

Public Attributes inherited from `Fl_RGB_Image`

- `int alloc_array`
  If non-zero, the object’s data array is `delete[]`d when deleting the object.

- `const uchar * array`
  Points to the start of the object’s data array.

Static Public Attributes inherited from `Fl_Image`

- `static const int ERR_FILE_ACCESS = -2`
- `static const int ERR_FORMAT = -3`
- `static const int ERR_MEMORY_ACCESS = -4`
- `static const int ERR_NO_IMAGE = -1`
- `static bool register_images_done = false`
  True after `fl_register_images()` was called, false before.

Protected Member Functions inherited from `Fl_Image`

- `void d (int D)`
  Sets the current image depth.

- `void data (const char *const *p, int c)`
  Sets the current data pointer and count of pointers in the array.

- `void draw_empty (int X, int Y)`
The protected method `draw_empty()` draws a box with an X in it.

- `int draw_scaled(int X, int Y, int W, int H)`
  Draw the image to the current drawing surface rescaled to a given width and height.
- `void h(int H)`
  Sets the height of the image data.
- `void ld(int LD)`
  Sets the current line data size in bytes.
- `void w(int W)`
  Sets the width of the image data.

### Static Protected Member Functions inherited from Fl_Image

- `static void labeltype(const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)`
- `static void measure(const Fl_Label *lo, int &lw, int &lh)`

#### 12.101.1 Detailed Description

The `Fl_PNG_Image` class supports loading, caching, and drawing of Portable Network Graphics (PNG) image files. The class loads color-mapped and full-color images and handles color- and alpha-based transparency.

#### 12.101.2 Constructor & Destructor Documentation

**12.101.2.1 Fl_PNG_Image()** [1/2]

```cpp
Fl_PNG_Image::Fl_PNG_Image (const char *filename)
```

The constructor loads the named PNG image from the given png filename. The destructor frees all memory and server resources that are used by the image. Use `Fl_PNG_Image::fail()` to check if `Fl_PNG_Image` failed to load. `fail()` returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the PNG format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filename</code></td>
<td>Name of PNG file to read</td>
</tr>
</tbody>
</table>

**12.101.2.2 Fl_PNG_Image()** [2/2]

```cpp
Fl_PNG_Image::Fl_PNG_Image (const char *name_png, const unsigned char *buffer, int maxsize)
```

Constructor that reads a PNG image from memory. Construct an image from a block of memory inside the application. Fluid offers "binary Data" chunks as a great way to add image data into the C++ source code. `name_png` can be NULL. If a name is given, the image is added to the list of shared images (see: `Fl_Shared_Image`) and will be available by that name.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name_png</code></td>
<td>A name given to this image or NULL</td>
</tr>
<tr>
<td><code>buffer</code></td>
<td>Pointer to the start of the PNG image in memory</td>
</tr>
<tr>
<td><code>maxsize</code></td>
<td>Size in bytes of the memory buffer containing the PNG image</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- `Fl_PNG_Image.H`
12.102  **Fl_PNM_Image** Class Reference

The **Fl_PNM_Image** class supports loading, caching, and drawing of Portable Anymap (PNM, PBM, PGM, PPM) image files.

```cpp
#include <Fl_PNM_Image.H>
```

Inheritance diagram for Fl_PNM_Image:

```
Fl_PNM_Image -> Fl_RGB_Image -> Fl_Image
```

**Public Member Functions**

- **Fl_PNM_Image** (const char ∗filename)
  
  The constructor loads the named PNM image.

**Public Member Functions inherited from Fl_RGB_Image**

- virtual **Fl_SVG_Image** ∗as_svg_image ()
  
  Returns whether an image is an Fl_SVG_Image or not.

- int cache_h ()

- int cache_w ()

- void color_average (Fl_Color c, float i) FL_OVERRIDE
  
  The color_average() method averages the colors in the image with the provided FLTK color value.

- **Fl_Image** ∗copy () const

- **Fl_Image** ∗copy (int W, int H) const FL_OVERRIDE
  
  Creates a resized copy of the image.

- void desaturate () FL_OVERRIDE
  
  The desaturate() method converts an image to grayscale.

- void draw (int X, int Y)

- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE
  
  Draws the image to the current drawing surface with a bounding box.

- **Fl_RGB_Image** (const Fl_Pixmap ∗pxm, Fl_Color bg=FL_GRAY)
  
  The constructor creates a new RGBA image from the specified Fl_Pixmap.

- **Fl_RGB_Image** (const uchar ∗bits, int bits_length, int W, int H, int D, int LD)

- **Fl_RGB_Image** (const uchar ∗bits, int W, int H, int D=3, int LD=0)

- void label (Fl_Menu_Item ∗m) FL_OVERRIDE
  
  This method is an obsolete way to set the image attribute of a menu item.

- void label (Fl_Widget ∗w) FL_OVERRIDE
  
  This method is an obsolete way to set the image attribute of a widget or menu item.

- virtual void normalize ()

  Makes sure the object is fully initialized.

- void uncache () FL_OVERRIDE
  
  If the image has been cached for display, delete the cache data.

- ~**Fl_RGB_Image** () FL_OVERRIDE

  The destructor frees all memory and server resources that are used by the image.
Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image ∗ as_shared_image ()
  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image ∗ copy () const
  Creates a copy of the image in the same size.
- int count () const
  Returns the number of data values associated with the image.
- int d () const
  Returns the image depth.
- const char ∗ const ∗ data () const
  Returns a pointer to the current image data array.
- int data_h () const
  Returns the height of the image data.
- int data_w () const
  Returns the width of the image data.
- void draw (int X, int Y)
  Draws the image to the current drawing surface.
- int fail () const
  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)
  The constructor creates an empty image with the specified width, height, and depth.
- int h () const
  Returns the current image drawing height in FLTK units.
- void inactive ()
  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- int ld () const
  Returns the current line data size in bytes.
- virtual void release ()
  Releases an Fl_Image - the same as 'delete this'.
- virtual void scale (int width, int height, int proportional=1, int can_expand=0)
  Sets the drawing size of the image.
- int w () const
  Returns the current image drawing width in FLTK units.
- virtual ~Fl_Image ()
  The destructor is a virtual method that frees all memory used by the image.

Additional Inherited Members

Static Public Member Functions inherited from Fl_RGB_Image

- static size_t max_size ()
  Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
- static void max_size (size_t size)
  Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype define_FL_IMAGE_LABEL ()
- static Fl_RGB_Scaling RGB_scaling ()
  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)
  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling scaling_algorithm ()
  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)
  Sets what algorithm is used when resizing a source image to draw it.

Public Attributes inherited from Fl_RGB_Image

- int alloc_array
  If non-zero, the object’s data array is delete[]’d when deleting the object.
- const uchar * array
  Points to the start of the object’s data array.

Static Public Attributes inherited from Fl_Image

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_MEMORY_ACCESS = -4
- static const int ERR_NO_IMAGE = -1
- static bool register_images_done = false
  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_Image

- void d (int D)
  Sets the current image depth.
- void data (const char *const *p, int c)
  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)
  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)
  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)
  Sets the height of the image data.
- void ld (int LD)
  Sets the current line data size in bytes.
- void w (int W)
  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void labeltype (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void measure (const Fl_Label *lo, int &lw, int &lh)

12.102.1 Detailed Description

The Fl_PNM_Image class supports loading, caching, and drawing of Portable Anymap (PNM, PBM, PGM, PPM) image files.
The class loads bitmap, grayscale, and full-color images in both ASCII and binary formats.
12.102.2 Constructor & Destructor Documentation

12.102.2.1 Fl_PNM_Image()

Fl_PNM_Image::Fl_PNM_Image (const char * filename )

The constructor loads the named PNM image. The destructor frees all memory and server resources that are used by the image. Use Fl_Image::fail() to check if Fl_PNM_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the PNM format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

**in** filename a full path and name pointing to a valid jpeg file.

The documentation for this class was generated from the following files:

- Fl_PNM_Image.H
- Fl_PNM_Image.cxx

12.103 Fl_Positioner Class Reference

This class is provided for Forms compatibility.

```
#include <Fl_Positioner.H>
```

Inheritance diagram for Fl_Positioner:

```
Fl_Widget
    ▼
     ▼
Fl_Positioner
```

Public Member Functions

- **Fl_Positioner** (int x, int y, int w, int h, const char *l=0)
  
  Creates a new Fl_Positioner widget using the given position, size, and label string.

- int handle (int) FL_OVERRIDE
  
  Handles the specified event.

- int value (double, double)
  
  Returns the current position in x and y.

- void xbounds (double, double)
  
  Sets the X axis bounds.

- double xmaximum () const
  
  Gets the X axis maximum.

- void xmaximum (double a)
  
  Same as xbounds(xminimum(), a)

- double xminimum () const
  
  Gets the X axis minimum.

- void xminimum (double a)
  
  Same as xbounds(a, xmaximum())

- void xstep (double a)
  
  Sets the stepping value for the X axis.

- double xvalue () const
  
  Gets the X axis coordinate.
• int \texttt{xvalue} (double)
  
  \textit{Sets the X axis coordinate.}

• void \texttt{ybounds} (double, double)
  
  \textit{Sets the Y axis bounds.}

• double \texttt{ymaximum} () const
  
  \textit{Gets the Y axis maximum.}

• void \texttt{ymaximum} (double a)
  
  \textit{Same as ybounds(\texttt{yminimum}(), a)}

• double \texttt{yminimum} () const
  
  \textit{Gets the Y axis minimum.}

• void \texttt{yminimum} (double a)
  
  \textit{Same as ybounds(a, \texttt{ymaximum}())}

• void \texttt{ystep} (double a)
  
  \textit{Sets the stepping value for the Y axis.}

• double \texttt{yvalue} () const
  
  \textit{Gets the Y axis coordinate.}

• int \texttt{yvalue} (double)
  
  \textit{Sets the Y axis coordinate.}

\textbf{Public Member Functions inherited from Fl_Widget}

• void \texttt{_clear_fullscreen} ()

• void \texttt{_set_fullscreen} ()

• void \texttt{activate} ()
  
  \textit{Activates the widget.}

• unsigned int \texttt{active} () const
  
  \textit{Returns whether the widget is active.}

• int \texttt{active_r} () const
  
  \textit{Returns whether the widget and all of its parents are active.}

• \texttt{Fl_Align} \texttt{align} () const
  
  \textit{Gets the label alignment.}

• void \texttt{align} (\texttt{Fl_Align} alignment)
  
  \textit{Sets the label alignment.}

• long \texttt{argument} () const
  
  \textit{Gets the current user data (long) argument that is passed to the callback function.}

• void \texttt{argument} (long v)
  
  \textit{Sets the current user data (long) argument that is passed to the callback function.}

• virtual class \texttt{Fl_Gl_Window \* as_gl_window} ()
  
  \textit{Returns an \texttt{Fl_Gl_Window} pointer if this widget is an \texttt{Fl_Gl_Window}.}

• virtual class \texttt{Fl_Gl_Window const \* as_gl_window} () const
  
  \textit{Returns an \texttt{Fl_Gl_Window} pointer if this widget is an \texttt{Fl_Gl_Window}.}

• virtual \texttt{Fl_Group \* as_group} ()
  
  \textit{Returns an \texttt{Fl_Group} pointer if this widget is an \texttt{Fl_Group}.}

• virtual \texttt{Fl_Group const \* as_group} () const
  
  \textit{Returns an \texttt{Fl_Group} pointer if this widget is an \texttt{Fl_Group}.}

• virtual \texttt{Fl_Window \* as_window} ()
  
  \textit{Returns an \texttt{Fl_Window} pointer if this widget is an \texttt{Fl_Window}.}

• virtual \texttt{Fl_Window const \* as_window} () const
  
  \textit{Returns an \texttt{Fl_Window} pointer if this widget is an \texttt{Fl_Window}.}

• void \texttt{bind_deimage} (\texttt{Fl_Image \*img})
  
  \textit{Sets the image to use as part of the widget label when in the inactive state.}

• void \texttt{bind_deimage} (int f)
  
  \textit{Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.}

• void \texttt{bind_image} (\texttt{Fl_Image \*img})
Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- Fl_Boxtype box () const
  Gets the box type of the widget.

- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.

- Fl_Callback_p callback () const
  Gets the current callback function for the widget.

- void callback (Fl_Callback *cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.

- void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.

- unsigned int changed () const
  Checks if the widget value changed since the last callback.

- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  Clears or sets the damage flags.

- void clear_output ()
  Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.

- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

- Fl_Color color () const
  Gets the background color of the widget.

- void color (Fl_Color bg)
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

- Fl_Color color2 () const
  For back compatibility only.

- void color2 (unsigned a)
  For back compatibility only.

- int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.

- void copy_label (const char *new_label)
  Sets the current label.

- void copy_tooltip (const char *text)
  Sets the current tooltip text.

- uchar damage () const
  Returns non-zero if draw() needs to be called.
• void **damage (uchar c)**
  
  **Sets the damage bits for the widget.**

• void **damage (uchar c, int x, int y, int w, int h)**
  
  **Sets the damage bits for an area inside the widget.**

• int **damage_resize** (int, int, int, int)
  
  **Internal use only.**

• void **deactivate ()**
  
  **Deactivates the widget.**

• Fl_Image ∗ **deimage ()**
  
  **Gets the image that is used as part of the widget label when in the inactive state.**

• const Fl_Image ∗ **deimage () const**
  
  **Gets the image that is used as part of the widget label when in the inactive state.**

• void **deimage (Fl_Image &img)**
  
  **Sets the image to use as part of the widget label when in the inactive state.**

• void **deimage (Fl_Image ∗ img)**
  
  **Sets the image to use as part of the widget label when in the inactive state.**

• int **deimage_bound () const**
  
  **Returns whether the inactive image is managed by the widget.**

• void **do_callback (Fl_Callback_Reas on=FL_REASON_UNKNOWN)**
  
  **Calls the widget callback function with default arguments.**

• void **do_callback (Fl_Widget ∗widget, long arg, Fl_Callback_Reas on=FL_REASON_UNKNOWN)**
  
  **Calls the widget callback function with arbitrary arguments.**

• void **do_callback (Fl_Widget ∗widget, void ∗arg=0, Fl_Callback_Reas on=FL_REASON_UNKNOWN)**
  
  **Calls the widget callback function with arbitrary arguments.**

• void **draw_label (int, int, int, int, Fl_Align) const**
  
  **Draws the label in an arbitrary bounding box with an arbitrary alignment.**

• int **h () const**
  
  **Gets the widget height.**

• virtual void **hide ()**
  
  **Makes a widget invisible.**

• Fl_Image ∗ **image ()**
  
  **Gets the image that is used as part of the widget label when in the active state.**

• const Fl_Image ∗ **image () const**
  
  **Gets the image that is used as part of the widget label when in the active state.**

• void **image (Fl_Image &img)**
  
  **Sets the image to use as part of the widget label when in the active state.**

• void **image (Fl_Image ∗ img)**
  
  **Sets the image to use as part of the widget label when in the active state.**

• int **image_bound () const**
  
  **Returns whether the image is managed by the widget.**

• int **inside (const Fl_Widget ∗wgt) const**
  
  **Checks if this widget is a child of wgt.**

• int **is_label_copied () const**
  
  **Returns whether the current label was assigned with copy_label().**

• const char ∗ **label () const**
  
  **Gets the current label text.**

• void **label (const char ∗text)**
  
  **Sets the current label pointer.**

• void **label (Fl_Labeltype a, const char ∗b)**
  
  **Shortcut to set the label text and type in one call.**

• Fl_Color **labelcolor () const**
• void labelcolor (Fl_Color c)
  Gets the label color.
• Fl_Font labelfont () const
  Sets the font to use.
• void labelfont (Fl_Font f)
  Gets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group * p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses ' & ' to indicate shortcuts.
• void shortcut_label (int value)
  
  Sets whether the widget’s label uses ’&’ to indicate shortcuts.
• virtual void show ()
  
  Makes a widget visible.
• void size (int W, int H)
  
  Changes the size of the widget.
• int take_focus ()
  
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  
  Returns if the widget is able to take events.
• int test_shortcut ()
  
  Returns true if the widget’s label contains the entered ’&x’ shortcut.
• const char ∗ tooltip () const
  
  Gets the current tooltip text.
• void tooltip (const char ∗text)
  
  Sets the current tooltip text.
• Fl_Window ∗ top_window () const
  
  Returns a pointer to the top-level window for the widget.
• Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const
  
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  
  Gets the widget type.
• void type (uchar t)
  
  Sets the widget type.
• int use_accents_menu ()
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void ∗ user_data () const
  
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data ∗v, bool auto_free)
  
  Sets the user data for this widget.
• void user_data (void ∗v)
  
  Sets the user data for this widget.
• unsigned int visible () const
  
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  
  Modifies keyboard focus navigation.
• int visible_r () const
  
  Returns whether a widget and all its parents are visible.
• int w () const
  
  Gets the widget width.
• Fl_When when () const
  
  Returns the conditions under which the callback is called.
• void when (uchar i)
  
  Sets the flags used to decide when a callback is called.
• Fl_Window ∗ window () const
  
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  
  Gets the widget position in its window.
• int y () const
  
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  
  Destroys the widget.
Protected Member Functions

- void `draw () FL_OVERRIDE`
  Draws the widget.
- void `draw (int, int, int, int)`
- int `handle (int, int, int, int)`

Protected Member Functions inherited from Fl_Widget

- void `clear_flag (unsigned int c)`
  Clears a flag in the flags mask.
- void `draw_backdrop () const`
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void `draw_box () const`
  Draws the widget box according its box style.
- void `draw_box (Fl_Boxtype t, Fl_Color c) const`
  Draws a box of type t, of color c at the widget's position and size.
- void `draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const`
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void `draw_focus () const`
  Draws a focus rectangle around the widget.
- void `draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const`
  Draws a focus rectangle around the widget.
- void `draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const`
  Draws a focus box for the widget at the given position and size.
- void `draw_label () const`
  Draws the widget's label at the defined label position.
- void `draw_label (int, int, int, int) const`
  Draws the label in an arbitrary bounding box.
- void `Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)`
  Creates a widget at the given position and size.
- unsigned int `flags () const`
  Gets the widget flags mask.
- void `h (int v)`
  Internal use only.
- void `set_flag (unsigned int c)`
  Sets a flag in the flags mask.
- void `w (int v)`
  Internal use only.
- void `x (int v)`
  Internal use only.
- void `y (int v)`
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void `default_callback (Fl_Widget ∗widget, void ∗data)`
  The default callback for all widgets that don’t set a callback.
- static unsigned int `label_shortcut (const char ∗t)`
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int `test_shortcut (const char ∗t, const bool require_alt=false)`
  Returns true if the given text t contains the entered '&x' shortcut.
Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLKIT = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22, AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

flags possible values enumeration.

12.103.1 Detailed Description

This class is provided for Forms compatibility. It provides 2D input. It would be useful if this could be put atop another widget so that the crosshairs are on top, but this is not implemented. The color of the crosshairs is selection_color().

![Figure 12.35 Fl_Positioner](image)

12.103.2 Constructor & Destructor Documentation

12.103.2.1 Fl_Positioner()

Fl_Positioner::Fl_Positioner (int X, int Y, int W, int H, const char * l = 0)

Creates a new Fl_Positioner widget using the given position, size, and label string. The default boxtype is FL_NO_BOX.

12.103.3 Member Function Documentation

12.103.3.1 draw()

void Fl_Positioner::draw () [protected], [virtual]

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets. If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```c
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.
12.103.3.2 handle()

```cpp
int Fl_Positioner::handle (  
    int event ) [virtual]
```

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class.

In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn’t do anything with it.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>the kind of event received</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the event was not used or understood</td>
</tr>
<tr>
<td>1</td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>

See also

- Fl_Event

Reimplemented from Fl_Widget.

The documentation for this class was generated from the following files:

- Fl_Positioner.H
- Fl_Positioner.cxx

12.104 Fl_PostScript_File_Device Class Reference

To send graphical output to a PostScript file.

```cpp
#include <Fl_PostScript.H>
```

Inheritance diagram for Fl_PostScript_File_Device:

```
Fl_Surface_Device
  
Fl_Widget_Surface
  
Fl_Paged_Device
  
Fl_PostScript_File_Device
```

Public Member Functions

- int begin_job (FILE *ps_output, int pageCount=0, enum Fl_Paged_Device::Page_Format format=Fl_Paged_Device::A4, enum Fl_Paged_Device::Page_Layout layout=Fl_Paged_Device::PORTRAIT)
Begins the session where all graphics requests will go to FILE pointer.

- int \texttt{begin\_job} (int pagecount, int *from, int *to, char **perr\_message) \texttt{FL\_OVERRIDE}
  
  Don't use with this class.

- int \texttt{begin\_job} (int pagecount=0, enum \texttt{Fl\_Paged\_Device::Page\_Format} format=\texttt{Fl\_Paged\_Device::A4}, enum \texttt{Fl\_Paged\_Device::Page\_Layout} layout=\texttt{Fl\_Paged\_Device::PORTRAIT})

  Begins the session where all graphics requests will go to a local PostScript file.

- int \texttt{begin\_page} (void) \texttt{FL\_OVERRIDE}

  Begins a new printed page.

- void \texttt{close\_command} (\texttt{Fl\_PostScript\_Close\_Command} cmd)

  Sets the function \texttt{end\_job()} calls to close the file()

- void \texttt{end\_current} () \texttt{FL\_OVERRIDE}

  FLTKit calls this each time a surface ceases to be the current drawing surface.

- void \texttt{end\_job} (void) \texttt{FL\_OVERRIDE}

  Completes all PostScript output.

- int \texttt{end\_page} (void) \texttt{FL\_OVERRIDE}

  To be called at the end of each page.

- FILE \* \texttt{file} ()

  Returns the underlying FILE\* receiving all PostScript data.

- \texttt{Fl\_PostScript\_File\_Device} ()

  The constructor.

- void \texttt{margins} (int *left, int *top, int *right, int *bottom) \texttt{FL\_OVERRIDE}

  Computes the dimensions of margins that lie between the printable page area and the full page.

- void \texttt{origin} (int *x, int *y) \texttt{FL\_OVERRIDE}

  Computes the coordinates of the current origin of graphics functions.

- void \texttt{origin} (int x, int y) \texttt{FL\_OVERRIDE}

  Sets the position of the origin of graphics in the drawable part of the drawing surface.

- int \texttt{printable\_rect} (int *w, int *h) \texttt{FL\_OVERRIDE}

  Computes the width and height of the drawable area of the drawing surface.

- void \texttt{rotate} (float angle) \texttt{FL\_OVERRIDE}

  Rotates the graphics operations relatively to paper.

- void \texttt{scale} (float scale\_x, float scale\_y=0.) \texttt{FL\_OVERRIDE}

  Changes the scaling of page coordinates.

- void \texttt{set\_current} () \texttt{FL\_OVERRIDE}

  Make this surface the current drawing surface.

- int \texttt{start\_job} (FILE \*ps\_output, int pagecount=0, enum \texttt{Fl\_Paged\_Device::Page\_Format} format=\texttt{Fl\_Paged\_Device::A4}, enum \texttt{Fl\_Paged\_Device::Page\_Layout} layout=\texttt{Fl\_Paged\_Device::PORTRAIT})

  Synonym of \texttt{begin\_job()}.

- int \texttt{start\_job} (int pagecount=0, enum \texttt{Fl\_Paged\_Device::Page\_Format} format=\texttt{Fl\_Paged\_Device::A4}, enum \texttt{Fl\_Paged\_Device::Page\_Layout} layout=\texttt{Fl\_Paged\_Device::PORTRAIT})

  Synonym of \texttt{begin\_job()}.

- void \texttt{translate} (int x, int y) \texttt{FL\_OVERRIDE}

  Translates the current graphics origin accounting for the current rotation.

- void \texttt{untranslate} (void) \texttt{FL\_OVERRIDE}

  Undoes the effect of a previous \texttt{translate()} call.

- \texttt{\sim\_Fl\_PostScript\_File\_Device} ()

  The destructor.
Public Member Functions inherited from Fl_Paged_Device

• void print_widget (Fl_Widget *widget, int delta_x=0, int delta_y=0)
  
  Synonym of draw(Fl_Widget*, int, int)

• void print_window (Fl_Window *win, int x_offset=0, int y_offset=0)
  
  Synonym of draw_decorated_window(Fl_Window*, int, int)

• int start_job (int pagecount=0, int *frompage=NULL, int *topage=NULL, char **perr_message=NULL)
  
  Synonym of begin_job(int pagecount, int *frompage, int *topage, char **perr_message).

• int start_page ()
  
  Synonym of begin_page().

• virtual ~Fl_Paged_Device ()
  
  The destructor.

Public Member Functions inherited from Fl_Widget_Surface

• void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)
  
  Draws the widget on the drawing surface.

• void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)
  
  Draws a window with its title bar and frame if any.

• void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)
  
  Draws a rectangular part of an on-screen window.

Public Member Functions inherited from Fl_Surface_Device

• Fl_Graphics_Driver * driver ()
  
  Returns the graphics driver of this drawing surface.

• virtual bool is_current ()
  
  Is this surface the current drawing surface?

• virtual ~Fl_Surface_Device ()
  
  The destructor.

Static Public Attributes

• static const char * file_chooser_title
  
  Label of the PostScript file chooser window.

Static Public Attributes inherited from Fl_Paged_Device

• static const page_format page_formats [NO_PAGE_FORMATS]
  
  width, height and name of all elements of the enum Page_Format.

Protected Member Functions

• Fl_PostScript_Graphics_Driver * driver ()
  
  Returns the PostScript driver of this drawing surface.

Protected Member Functions inherited from Fl_Paged_Device

• Fl_Paged_Device ()
  
  The constructor.

Protected Member Functions inherited from Fl_Widget_Surface

• Fl_Widget_Surface (Fl_Graphics_Driver *d)
  
  The constructor.
Protected Member Functions inherited from Fl_Surface_Device

- void driver (Fl_Graphics_Driver *graphics_driver)
  
  Sets the graphics driver of this drawing surface.

- Fl_Surface_Device (Fl_Graphics_Driver *graphics_driver)
  
  Constructor that sets the graphics driver to use for the created surface.

Additional Inherited Members

Public Types inherited from Fl_Paged_Device

- enum Page_Format { A0 = 0 , A1 , A2 , A3 , A4 , A5 , A6 , A7 , A8 , A9 , B0 , B1 , B2 , B3 , B4 , B5 , B6 , B7 , B8 , B9 , B10 , C5E , DLE , EXECUTIVE , FOLIO , LEDGER , LEGAL , LETTER , TABLOID , ENVELOPE , MEDIA = 0x1000 }  
  
  Possible page formats.

- enum Page_Layout { PORTRAIT = 0 , LANDSCAPE = 0x100 , REVERSED = 0x200 , ORIENTATION = 0x300 }
  
  Possible page layouts.

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()
  
  Removes the top element from the current drawing surface stack, and makes the new top element current.

- static void push_current (Fl_Surface_Device *new_current)
  
  Pushes new_current on top of the stack of current drawing surfaces, and makes it current.

- static Fl_Surface_Device * surface ()
  
  The current drawing surface.

Protected Attributes inherited from Fl_Widget_Surface

- int x_offset  
  
  Horizontal offset to the origin of graphics coordinates.

- int y_offset  
  
  Vertical offset to the origin of graphics coordinates.

12.104.1 Detailed Description

To send graphical output to a PostScript file.
This class is used exactly as the Fl_Printer class except for the begin_job() call, two variants of which are usable and allow to specify what page format and layout are desired.

Processing of text: Text uses vectorial fonts under the X11 + Pango and the Wayland platforms. With other platforms, only text restricted to the Latin alphabet (and a few other characters listed in the table below) and to FLTK standard fonts is vectorized. All other unicode characters or all other fonts (FL_FREE_FONT and above) are output as a bitmap. FLTK standard fonts are output using the corresponding PostScript standard fonts. The latin alphabet means all unicode characters between U+0020 and U+017F, or, in other words, the ASCII, Latin-1 Supplement and Latin Extended-A charts.
12.104 Fl_PostScript_File_Device Class Reference

12.104.2 Member Function Documentation

12.104.2.1 begin_job() [1/3]

```c
int Fl_PostScript_File_Device::begin_job ( 
    FILE * ps_output, 
    int pagecount = 0, 
    enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4, 
    enum Fl_Paged_Device::Page_Layer layout = Fl_Paged_Device::PORTRAIT )
```

Begins the session where all graphics requests will go to FILE pointer.
This member function prevents `end_job()` from closing `ps_output`, so the user can check with `ferror(ps_←output)` for output errors.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ps_output</code></td>
<td>A writable FILE pointer that will receive PostScript output and that should not be closed until after <code>end_job()</code> has been called.</td>
</tr>
<tr>
<td><code>pagecount</code></td>
<td>The total number of pages to be created. Use 0 if this number is unknown when this function is called.</td>
</tr>
<tr>
<td><code>format</code></td>
<td>Desired page format.</td>
</tr>
<tr>
<td><code>layout</code></td>
<td>Desired page layout.</td>
</tr>
</tbody>
</table>

Returns

always 0.

12.104.2.2 begin_job() [2/3]

```c
int Fl_PostScript_File_Device::begin_job ( 
    int pagecount, 
    int * from,
```

Figure 12.36 Extra characters supported by standard PostScript fonts

Processing of transparent Fl_RGB_Image objects: Under the X11 + Pango and the Wayland platforms, these objects are output with their exact transparency. With other platforms, these objects are drawn blended to white color. Class Fl_EPS_File_Surface’s constructor allows to set another background color for blending.

<table>
<thead>
<tr>
<th>Char</th>
<th>Codepoint</th>
<th>Name</th>
<th>Char</th>
<th>Codepoint</th>
<th>Name</th>
<th>Char</th>
<th>Codepoint</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>U+0192</td>
<td>florin</td>
<td>·</td>
<td>U+201A</td>
<td>quotingbase</td>
<td>™</td>
<td>U+2122</td>
<td>trademark</td>
</tr>
<tr>
<td>&quot;</td>
<td>U+02C6</td>
<td>circumflex</td>
<td>&quot;</td>
<td>U+201C</td>
<td>quotedblleft</td>
<td>å</td>
<td>U+2202</td>
<td>partialdiff</td>
</tr>
<tr>
<td>&quot;</td>
<td>U+02C7</td>
<td>caron</td>
<td>&quot;</td>
<td>U+201D</td>
<td>quotedblright</td>
<td>∆</td>
<td>U+2206</td>
<td>Delta</td>
</tr>
<tr>
<td>&quot;</td>
<td>U+02D8</td>
<td>breve</td>
<td>&quot;</td>
<td>U+201E</td>
<td>quotedblbase</td>
<td>Σ</td>
<td>U+2211</td>
<td>summation</td>
</tr>
<tr>
<td>·</td>
<td>U+02D9</td>
<td>dotaccent</td>
<td>†</td>
<td>U+2020</td>
<td>dagger</td>
<td>√</td>
<td>U+221A</td>
<td>radical</td>
</tr>
<tr>
<td>·</td>
<td>U+02DA</td>
<td>ring</td>
<td>††</td>
<td>U+2021</td>
<td>daggerdbl</td>
<td>∞</td>
<td>U+221E</td>
<td>infinity</td>
</tr>
<tr>
<td>·</td>
<td>U+02DB</td>
<td>ogonek</td>
<td>·</td>
<td>U+2022</td>
<td>bullet</td>
<td>≠</td>
<td>U+2260</td>
<td>notequal</td>
</tr>
<tr>
<td>·</td>
<td>U+02DC</td>
<td>tildel</td>
<td>...</td>
<td>U+2026</td>
<td>ellipsis</td>
<td>≦</td>
<td>U+2264</td>
<td>lessequal</td>
</tr>
<tr>
<td>·</td>
<td>U+02DD</td>
<td>hungarumlaut</td>
<td>%</td>
<td>U+2030</td>
<td>perthousand</td>
<td>≻</td>
<td>U+2265</td>
<td>greaterequal</td>
</tr>
<tr>
<td>—</td>
<td>U+2013</td>
<td>endash</td>
<td>'</td>
<td>U+2039</td>
<td>guillemotleft</td>
<td>°</td>
<td>U+25CA</td>
<td>lozenge</td>
</tr>
<tr>
<td>—</td>
<td>U+2014</td>
<td>emdash</td>
<td>'</td>
<td>U+203A</td>
<td>guillemotright</td>
<td>fi</td>
<td>U+FB01</td>
<td>fi</td>
</tr>
<tr>
<td>'</td>
<td>U+2018</td>
<td>quotell</td>
<td>/</td>
<td>U+2044</td>
<td>fraction</td>
<td>fl</td>
<td>U+FB02</td>
<td>fi</td>
</tr>
<tr>
<td>'</td>
<td>U+2019</td>
<td>quoteright</td>
<td>€</td>
<td>U+20AC</td>
<td>Euro</td>
<td>♠</td>
<td>U+F8FF</td>
<td>apple (macOS only)</td>
</tr>
</tbody>
</table>
int * to,
char ** perr_message ) [virtual]

Don't use with this class.
Reimplemented from Fl_Paged_Device.

12.104.2.3 begin_job() [3/3]

int Fl_PostScript_File_Device::begin_job (  
    int pagecount = 0,
    enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
    enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT )

Begins the session where all graphics requests will go to a local PostScript file.
Opens a file dialog to select an output PostScript file. This member function makes end_job() close the resulting
PostScript file and display an alert message with fl_alert() in case of any output error.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pagecount</td>
<td>The total number of pages to be created. Use 0 if this number is unknown when this function is called.</td>
</tr>
<tr>
<td>format</td>
<td>Desired page format.</td>
</tr>
<tr>
<td>layout</td>
<td>Desired page layout.</td>
</tr>
</tbody>
</table>

Returns

0 if OK, 1 if user cancelled the file dialog, 2 if fopen failed on user-selected output file.

12.104.2.4 begin_page()

int Fl_PostScript_File_Device::begin_page (  
    void ) [virtual]

Begins a new printed page.
The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area.
This function also makes this surface the current drawing surface with Fl_Surface_Device::push_current().

Note

begin_page() calls Fl_Surface_Device::push_current() and leaves this device as the active surface. If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current(), or the content of the dialog box will be rendered to the printer instead of the screen.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Paged_Device.

12.104.2.5 end_current()

void Fl_PostScript_File_Device::end_current ( ) [virtual]

FLTK calls this each time a surface ceases to be the current drawing surface.
This member function is mostly of interest to developers of new Fl_Surface_Device derived classes. It allows to perform surface-specific operations necessary when this surface ceases to be current. Each implementation should end with a call to Fl_Surface_Device::end_current().
Reimplemented from Fl_Surface_Device.
12.104.2.6 end_job()

```cpp
void Fl_PostScript_File_Device::end_job {
  void ) [virtual]
}
```

Completes all PostScript output.
This also closes with fclose() the underlying file() unless close_command() was used to set another function.
Reimplemented from Fl_Paged_Device.

12.104.2.7 end_page()

```cpp
int Fl_PostScript_File_Device::end_page {
  void ) [virtual]
}
```

To be called at the end of each page.
This function also stops this surface from being the current drawing surface with Fl_Surface_Device::pop_current().

Note

end_page() calls Fl_Surface_Device::pop_current(). If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to
Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current().

Returns

0 if OK, non-zero if any error.
Reimplemented from Fl_Paged_Device.

12.104.2.8 margins()

```cpp
void Fl_PostScript_File_Device::margins {
  int * left,
  int * top,
  int * right,
  int * bottom ) [virtual]
}
```

Computes the dimensions of margins that lie between the printable page area and the full page.
Values are in the same unit as that used by FLTK drawing functions. They are changed by scale() calls.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out left</code></td>
<td>If non-null, <code>*left</code> is set to the left margin size.</td>
</tr>
<tr>
<td><code>out top</code></td>
<td>If non-null, <code>*top</code> is set to the top margin size.</td>
</tr>
<tr>
<td><code>out right</code></td>
<td>If non-null, <code>*right</code> is set to the right margin size.</td>
</tr>
<tr>
<td><code>out bottom</code></td>
<td>If non-null, <code>*bottom</code> is set to the bottom margin size.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

12.104.2.9 origin() [1/2]

```cpp
void Fl_PostScript_File_Device::origin {
  int * x,
  int * y ) [virtual]
}
```

Computes the coordinates of the current origin of graphics functions.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out x,y</code></td>
<td>If non-null, <code>*x</code> and <code>*y</code> are set to the horizontal and vertical coordinates of the graphics origin.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Widget_Surface.
12.104.2.10 origin() [2/2]

```cpp
void Fl_PostScript_File_Device::origin ( 
    int x, 
    int y ) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface. Arguments should be expressed relatively to the result of a previous `printable_rect()` call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive `origin()` calls don't combine their effects. `Origin()` calls are not affected by `rotate()` calls (for classes derived from `Fl_Paged_Device`).

**Parameters**

| in | x, y | Horizontal and vertical positions in the drawing surface of the desired origin of graphics. |

Reimplemented from `Fl_Widget_Surface`.

12.104.2.11 printable_rect()

```cpp
int Fl_PostScript_File_Device::printable_rect ( 
    int * w, 
    int * h ) [virtual]
```

Computes the width and height of the drawable area of the drawing surface. Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to `origin()`. If the object is derived from class `Fl_Paged_Device`, values account for the user-selected paper type and print orientation and are changed by `scale()` calls.

**Returns**

0 if OK, non-zero if any error

Reimplemented from `Fl_Widget_Surface`.

12.104.2.12 rotate()

```cpp
void Fl_PostScript_File_Device::rotate ( 
    float angle ) [virtual]
```

Rotates the graphics operations relatively to paper. The rotation is centered on the current graphics origin. Successive `rotate()` calls don't combine their effects.

**Parameters**

| angle | Rotation angle in counter-clockwise degrees. |

Reimplemented from `Fl_Paged_Device`.

12.104.2.13 scale()

```cpp
void Fl_PostScript_File_Device::scale ( 
    float scale_x, 
    float scale_y = 0. ) [virtual]
```

Changes the scaling of page coordinates. This function also resets the origin of graphics functions at top left of printable page area. After a `scale()` call, do a `printable_rect()` call to get the new dimensions of the printable page area. Successive `scale()` calls don't combine their effects.

**Parameters**

| scale←_x | Horizontal dimensions of plot are multiplied by this quantity. |
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>scale_y</em></td>
<td>Same as above, vertically. The value 0. is equivalent to setting scale_y = scale_x. Thus, scale(factor); is equivalent to scale(factor, factor);</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

12.104.2.14 set_current()

```cpp
void Fl_PostScript_File_Device::set_current ( ) [virtual]
```

Make this surface the current drawing surface. This surface will receive all future graphics requests. Starting from FLTK 1.4.0, the preferred API to change the current drawing surface is Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current().

**Note**

It's recommended to use this function only as follows:

- The current drawing surface is the display;
- make current another surface, e.g., an Fl_Printer or an Fl_Image_Surface object, calling set_current() on this object;
- draw to that surface;
- make the display current again with Fl_Display_Device::display_device()->set_current();. Don't do any other call to set_current() before this one.

Other scenarios of drawing surface changes should be performed via Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current().

Reimplemented from Fl_Surface_Device.

12.104.2.15 start_job() [1/2]

```cpp
int Fl_PostScript_File_Device::start_job ( FILE * ps_output, int pagecount = 0, enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4, enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT ) [inline]
```

Synonym of begin_job().

For API compatibility with FLTK 1.3.x

12.104.2.16 start_job() [2/2]

```cpp
int Fl_PostScript_File_Device::start_job ( int pagecount = 0, enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4, enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT ) [inline]
```

Synonym of begin_job().

For API compatibility with FLTK 1.3.x

12.104.2.17 translate()

```cpp
void Fl_PostScript_File_Device::translate ( int x, int y ) [virtual]
```

Translates the current graphics origin accounting for the current rotation. Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects. Reimplemented from Fl_Widget_Surface.
12.104.2.18 untranslate()

`void Fl_PostScript_File_Device::untranslate ( void )` [virtual]

Undoes the effect of a previous `translate()` call.

Reimplemented from `Fl_Widget_Surface`.

The documentation for this class was generated from the following file:

- `Fl_PostScript.H`

12.105 Fl_Preferences Class Reference

`Fl_Preferences` store user settings between application starts.

```cpp
#include <Fl_Preferences.H>
```

Inheritance diagram for `Fl_Preferences`:

```
    Fl_Preferences
    |   
    |   Fl_Plugin_Manager
```

Classes

- struct Entry
- class Name

  'Name' provides a simple method to create numerical or more complex procedural names for entries and groups on the fly.

- class Node
- class RootNode

Public Types

- typedef void * ID

  Every `Fl_Preferences`-Group has a unique ID.

- enum Root {
  UNKNOWN_ROOT_TYPE = -1 , SYSTEM = 0 , USER , MEMORY ,
  ROOT_MASK = 0x00FF , CORE = 0x0100 , C_LOCALE = 0x1000 , CLEAR = 0x2000 ,
  SYSTEM_L = SYSTEM | C_LOCALE , USER_L = USER | C_LOCALE , CORE_SYSTEM_L = CORE | SYSTEM_L , CORE_USER_L = CORE | USER_L ,
  CORE_SYSTEM = CORE | SYSTEM , CORE_USER = CORE | USER }

  Define the scope of the preferences.

Public Member Functions

- char `clear ()`

  Delete all groups and all entries.

- char `delete_all_entries ()`

  Delete all entries.

- char `delete_all_groups ()`

  Delete all groups.

- char `delete_entry (const char *entry)`

  Deletes a single name/value pair.

- char `delete_group (const char *group)`

  Deletes a group.

- int `dirty ()`
Check if there were changes to the database that need to be written to disk.

- int entries()
  Returns the number of entries (name/value pairs) in a group.
- const char * entry (int index)
  Returns the name of an entry.
- char entry_exists (const char *key)
  Returns non-zero if an entry with this name exists.
- Root filename (char *buffer, size_t buffer_size)
  Return the file name and path to the preference file.
- Fl_Preferences (const char *path, const char *vendor, const char *application)
  Deprecated: Use this constructor to create or read a preference file at an arbitrary position in the file system.
- Fl_Preferences (const char *path, const char *vendor, const char *application, Root flags)
  Use this constructor to create or read a preference file at an arbitrary position in the file system.
- Fl_Preferences (const Fl_Preferences &)
  Create another reference to a Preferences group.
- Fl_Preferences (Fl_Preferences &parent, const char *group)
  Generate or read a new group of entries within another group.
- Fl_Preferences (Fl_Preferences &parent, int groupIndex)
  Open a child group using a given index.
- Fl_Preferences (Fl_Preferences *parent, const char *group)
  Create or access a group of preferences using a name.
- Fl_Preferences (Fl_Preferences *parent, int groupIndex)
- Fl_Preferences (ID id)
  Create a new dataset access point using a dataset ID.
- Fl_Preferences (Root root, const char *vendor, const char *application)
  The constructor creates a group that manages key/value pairs and child groups.
- int flush()
  Writes preferences to disk if they were modified.
- char get (const char *entry, char *&value, const char *defaultValue)
  Reads an entry from the group.
- char get (const char *entry, char *value, const char *defaultValue, int maxSize)
  Reads an entry from the group.
- char get (const char *entry, double &value, double defaultValue)
  Reads an entry from the group.
- char get (const char *entry, float &value, float defaultValue)
  Reads an entry from the group.
- char get (const char *entry, int &value, int defaultValue)
  Reads an entry from the group.
- char get (const char *entry, void *&value, const void *defaultValue, int defaultSize)
  Reads an entry from the group.
- char get (const char *entry, void *value, const void *defaultValue, int defaultSize, int *size)
  Reads a binary entry from the group, encoded in hexadecimal blocks.
- char get (const char *entry, void *value, const void *defaultValue, int defaultSize, int maxSize)
  Reads a binary entry from the group, encoded in hexadecimal blocks.
- char get_userdata_path (char *path, int pathlen)
  Creates a path that is related to the preference file and that is usable for additional application data.
- const char * group (int num_group)
  Returns the name of the Nth (num_group) group.
- char group_exists (const char *key)
  Returns non-zero if a group with this name exists.
- int groups ()
Returns the number of groups that are contained within a group.

- **ID id ()**
  Return an ID that can later be reused to open more references to this dataset.

- **const char * name ()**
  Return the name of this entry.

- **const char * path ()**
  Return the full path to this entry.

- **char set (const char * entry, const char *value)**
  Sets an entry (name/value pair).

- **char set (const char * entry, const void *value, int size)**
  Sets an entry (name/value pair).

- **char set (const char * entry, double value)**
  Sets an entry (name/value pair).

- **char set (const char * entry, double value, int precision)**
  Sets an entry (name/value pair).

- **char set (const char * entry, float value)**
  Sets an entry (name/value pair).

- **char set (const char * entry, float value, int precision)**
  Sets an entry (name/value pair).

- **char set (const char * entry, int value)**
  Sets an entry (name/value pair).

- **int size (const char *entry)**
  Returns the size of the value part of an entry.

- **virtual ~FL_Preferences ()**
  The destructor removes allocated resources.

### Static Public Member Functions

- **static unsigned int file_access ()**
  Return the current file access permissions for the FLTK preferences system.

- **static void file_access (unsigned int flags)**
  Tell the FLTK preferences system which files in the file system it may read, create, or write.

- **static Root filename (char *buffer, size_t buffer_size, Root root, const char *vendor, const char *application)**
  Determine the file name and path to preferences that would be opened with these parameters.

- **static const char *new_UUID ()**
  Returns a UUID as generated by the system.

- **static char remove (ID id_)**
  Remove the group with this ID from a database.

### Static Public Attributes

- **static const unsigned int ALL = ALL_READ_OK | ALL_WRITE_OK**
  Set this to give FLTK and applications permission to read, write, and create preference files.

- **static const unsigned int ALL_READ_OK = USER_READ_OK | SYSTEM_READ_OK | CORE_READ_OK**
  Set this to allow FLTK and applications to read preference files.

- **static const unsigned int ALL_WRITE_OK = USER_WRITE_OK | SYSTEM_WRITE_OK | CORE_WRITE_OK**
  Set this to allow FLTK and applications to create and write preference files.

- **static const unsigned int APP_OK = SYSTEM_OK | USER_OK**
  Set this if it is OK for applications to read, create, and write any kind of preference files.

- **static const unsigned int CORE_OK = CORE_READ_OK | CORE_WRITE_OK**
  Set this if it is OK for FLTK to read, create, or write preference files.

- **static const unsigned int CORE_READ_OK = 0x0010**
Set this if it is OK for FLTK to read preference files.
- static const unsigned int CORE_WRITE_OK = 0x0020
  Set this if it is OK for FLTK to create or write preference files.
- static const unsigned int NONE = 0x0000
  Set this if no call to Fl_Preferences shall access the file system.
- static const unsigned int SYSTEM_OK = SYSTEM_READ_OK | SYSTEM_WRITE_OK
  Set this if it is OK for applications to read, create, and write system wide preference files.
- static const unsigned int SYSTEM_READ_OK = 0x0004
  Set this if it is OK for applications to read system wide preference files.
- static const unsigned int SYSTEM_WRITE_OK = 0x0008
  Set this if it is OK for applications to create and write system wide preference files.
- static const unsigned int USER_OK = USER_READ_OK | USER_WRITE_OK
  Set this if it is OK for applications to read, create, and write user preference files.
- static const unsigned int USER_READ_OK = 0x0001
  Set this if it is OK for applications to read user preference files.
- static const unsigned int USER_WRITE_OK = 0x0002
  Set this if it is OK for applications to create and write user preference files.

Protected Attributes
- Node * node
- RootNode * rootNode

Friends
- class Node
- class RootNode

12.105.1 Detailed Description

Fl_Preferences store user settings between application starts. Fl_Preferences are similar to the Registry on Windows and Preferences on MacOS, providing a simple method to store customizable user settings between app launches, for instance the previous window position or a history of previously used documents.

Preferences are organized in a hierarchy of groups. Every group can contain more groups and any number of key/value pairs. Keys can be text strings containing ASCII letters, digits, periods, and underscores. Forward slashes in a key name are treated as subgroups, i.e. the key 'window/width' would actually refer to the key 'width' inside the group 'window'.

Keys usually have a unique name within their group. Duplicate keys are possible though and can be accessed using the index based functions.

A value can be an UTF-8 string. Control characters and UTF-8 sequences are stored as octal values. Long strings are wrapped at the line ending and will be reassembled when reading the file back.

Several methods allow setting and getting numerical values and binary data.

Preferences are stored in text files that can be edited manually if needed. The file format is easy to read and relatively forgiving. Preference files are the same on all platforms. User comments in preference files are preserved. Filenames are unique for each application by using a vendor/application naming scheme. The user must provide default values for all entries to ensure proper operation should preferences be corrupted or not yet exist.

FLTK preferences are not meant to replace a fully features database. No merging of data takes place. If several instances of an app access the same database at the same time, only the most recent changes will persist.

Preferences should not be used to store document data. The .prefs file should be kept small for performance reasons. One application can have multiple preference files. Extensive binary data however should be stored in separate files: see Fl_Preferences::get_userdata_path().

Fl_Preferences are not thread-safe. They can temporarily change the locale on some platforms during read and write access, which also changes it temporarily in other threads of the same app.

Typically a preferences database is read at startup, and then reopened and written at app shutdown.
int appWindowWidth, appWindowHeight;

void launch() {
    FL_Preferences app(FL_PREFERENCES::USER_L, "matthiasm.com", "hello");
    // 'app' constructor will be called, reading data from .prefs file
    FL_PREFERENCES window(app, "window");
    window.get("width", appWindowWidth, 800);
    window.get("height", appWindowHeight, 600);
    // 'app' destructor will be called. This will write data to the
    // .prefs file if any preferences were changed or added
}

void quit() {
    FL_PREFERENCES app(FL_PREFERENCES::USER_L, "matthiasm.com", "hello");
    FL_PREFERENCES window(app, "window");
    window.set("width", appWindowWidth);
    window.set("height", appWindowHeight);
}

See also

    FL_PREFERENCES::FL_PREFERENCES(Root root, const char ∗vendor, const char ∗application)

As a special case, FL_PREFERENCES can be memory mapped and not be associated with a file on disk.

See also

    FL_PREFERENCES::FL_PREFERENCES(FL_PREFERENCES ∗parent, const char ∗group) for more details on memory
    mapped preferences.

Note

Starting with FLTK 1.3, preference databases are expected to be in UTF-8 encoding. Previous databases
were stored in the current character set or code page which renders them incompatible for text entries using
international characters.

Starting with FLTK 1.4, searching a valid path to store the preference files has changed slightly. Please see
FL_PREFERENCES::FL_PREFERENCES(Root, const char ∗, const char ∗) for details.

Starting with FLTK 1.4, preference files should be created with SYSTEM_L or USER_L to be interchangeable
between computers with differing locale settings. The legacy modes, LOCAL and SYSTEM, will read and
write floating point values using the decimal point of the current locale. As a result, a fp-value would be written
‘3,1415’ on a German machine, and would be read back as ‘3.0’ on a US machine because the comma would
not be recognized as an alternative decimal point.

12.105.2 Member Typedef Documentation

12.105.2.1 ID

typedef void ∗FL_PREFERENCES::ID

Every FL_PREFERENCES-Group has a unique ID. ID's can be retrieved from an FL_PREFERENCES-Group and can then be used to create more FL_Preference references
to the same data set, as long as the database remains open.

12.105.3 Member Enumeration Documentation

12.105.3.1 Root

enum FL_PREFERENCES::Root

Define the scope of the preferences.

Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNKNOWN_ROOT_TYPE</td>
<td>Returned if storage could not be determined.</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Preferences are used system-wide. Deprecated, see SYSTEM_L.</td>
</tr>
<tr>
<td>USER</td>
<td>Preferences apply only to the current user. Deprecated, see USER_L.</td>
</tr>
<tr>
<td>MEMORY</td>
<td>Returned if querying memory mapped preferences.</td>
</tr>
<tr>
<td>ROOT_MASK</td>
<td>Mask for the values above.</td>
</tr>
</tbody>
</table>
12.105 Fl_Preferences Class Reference

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORE</td>
<td>OR'd by FLTK to read and write core library preferences and options.</td>
</tr>
<tr>
<td>C_LOCALE</td>
<td>This flag should always be set, it makes sure that floating point.</td>
</tr>
<tr>
<td>CLEAR</td>
<td>Don't read a possibly existing database. Instead, start with an empty set of preferences. Values are written correctly independently of the current locale.</td>
</tr>
<tr>
<td>SYSTEM_L</td>
<td>Preferences are used system-wide, locale independent.</td>
</tr>
<tr>
<td>USER_L</td>
<td>Preferences apply only to the current user, locale independent.</td>
</tr>
<tr>
<td>CORE_SYSTEM_L</td>
<td>Same as CORE</td>
</tr>
<tr>
<td>CORE_USER_L</td>
<td>Same as CORE</td>
</tr>
<tr>
<td>CORE_SYSTEM</td>
<td>Deprecated, same as CORE</td>
</tr>
<tr>
<td>CORE_USER</td>
<td>Deprecated, same as CORE</td>
</tr>
</tbody>
</table>

12.105.4 Constructor & Destructor Documentation

12.105.4.1 Fl_Preferences() [1/8]

Fl_Preferences::Fl_Preferences ( 
    Root root, 
    const char * vendor, 
    const char * application )  

The constructor creates a group that manages key/value pairs and child groups. Preferences can be stored per user using the root type Fl_Preferences::USER_L, or stored system-wide using Fl_Preferences::SYSTEM_L.

Groups and key/value pairs can be read and written randomly. Reading undefined values will return the default value. Writing undefined values will create all required groups and key/value pairs.

This constructor creates the base instance for all following entries and reads the database from disk into memory if it exists. The vendor argument is a unique text string identifying the development team or vendor of an application. A domain name or an EMail address (replacing the '@' with a '.') are great unique names, e.g. "research.matthiasm.com" or "fluid.fltk.org". The application argument can be the working title or final name of your application. Both vendor and application must be valid UNIX path segments as they become parts of the preference file path and may contain forward slashes to create deeper file structures.

Note

On Windows, the directory is constructed by querying the Common AppData or AppData key of the Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders registry entry. The filename and path is then constructed as $(query)/$(vendor)/$(application).prefs. If the query call fails, data will be stored in RAM only. It will be lost when the app exits.

In FLTK versions before 1.4.0, if querying the registry failed, preferences would be written to C:\FLTK\$(vendor)\$(application).prefs.

Note

On Linux, the USER directory is constructed by reading $HOME. If $HOME is not set or not pointing to an existing directory, FLTK will check the path member of the passwd struct returned by getpwuid(getuid()) . If all attempts fail, data will be stored in RAM only and be lost when the app exits.

The SYSTEM preference filename is hardcoded as /etc/fltk/$(vendor)/$(application).prefs. For backward compatibility, the old USER.prefs file naming scheme $(directory)/.fltk/$(vendor)/$(application).prefs is checked first. If that file does not exist, the environment variable $XDG_CONFIG_HOME is read as a base directory. If $XDG_CONFIG_HOME is not set, the base directory defaults to $HOME/.config/. The user preferences will be stored in $(directory)/$(vendor)/$(application).prefs. The user data path will be $(directory)/$(vendor)/$(application)/. In FLTK versions before 1.4.0, if $HOME was not set, the USER path would be empty, generating $(vendor)/$(application).prefs, which was used relative to the current working directory.
Note

On macOS, the USER directory is constructed by reading $HOME. If $HOME is not set or not pointing to an existing directory, we check the path returned by NSHomeDirectory, and finally checking the path member of the passwd struct returned by getpwuid(getuid). If all attempts fail, data will be stored in RAM only and be lost when the app exits. The filename and path is then constructed as $(directory)/~Library/Preferences/${vendor}/$(application).prefs. The SYSTEM directory is hard-coded as /Library/Preferences/${vendor}/$(application).prefs.

In FLTK versions before 1.4.0, if $HOME was not set, the USER path would be NULL, generating <null>/Library/Preferences/${vendor}/$(application).prefs, which would silently fail to create a preference file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>can be USER_L or SYSTEM_L for user specific or system wide preferences, add the CLEAR flag to start with a clean set of preferences instead of reading them from a preexisting database</td>
</tr>
<tr>
<td>vendor</td>
<td>unique text describing the company or author of this file, must be a valid filepath segment</td>
</tr>
<tr>
<td>application</td>
<td>unique text describing the application, must be a valid filepath segment</td>
</tr>
</tbody>
</table>

See also

Fl_Preferences(Fl_Preferences *parent, const char *group) with parent set to NULL

12.105.4.2 Fl_Preferences() [2/8]

Fl_Preferences::Fl_Preferences (  
const char * path,  
const char * vendor,  
const char * application,  
Root flags )  

Use this constructor to create or read a preference file at an arbitrary position in the file system. The file name is generated in the form $(path)/$(application).prefs. If application is NULL, path is taken literally as the file path and name.

// Example: read from an existing database and write modifications when flushed
Fl_Preferences database("/user/matt/test.prefs", "org.fltk.test", NULL, Fl_Preferences::C_LOCALE);

// Example: create a new preferences file with an empty data set
Fl_Preferences database("/user/matt/test.prefs", "org.fltk.test", NULL, Fl_Preferences::Root)(Fl_Preferences::C_LOCALE|Fl_Preferences::CLEAR));

Note

the C_LOCALE flag is is not set by default for backward compatibility, but it is highly recommended to set it when opening a database.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>path to the directory that contains the preference file</td>
</tr>
<tr>
<td>vendor</td>
<td>unique text describing the company or author of this file, must be a valid file path segment</td>
</tr>
<tr>
<td>application</td>
<td>unique text describing the application, must be a valid filename or NULL</td>
</tr>
<tr>
<td>flags</td>
<td>C_LOCALE to make the preferences file independent of the current locale, add the CLEAR flag to start with a clean set of preferences instead of reading from the database</td>
</tr>
</tbody>
</table>
12.105.3 Fl_Preferences() [3/8]

FL_PREFERENCES:

| Fl_Preferences & parent, |
| const char * group |

Generate or read a new group of entries within another group. Use the `group` argument to name the group that you would like to access. Group can also contain a path to a group further down the hierarchy by separating group names with a forward slash '/'.

Parameters

| in parent |
| reference object for the new group |
| in group |
| name of the group to access (may contain '/'s) |

12.105.4 Fl_Preferences() [4/8]

FL_PREFERENCES:

| Fl_Preferences & parent, |
| const char * group |

Create or access a group of preferences using a name. Parent should point to a previously created parent preferences group to create a preferences hierarchy. If `parent` is set to NULL, an unnamed database will be accessed that exists only in local memory and is not associated with a file on disk. The root type of this database is set to `Fl_Preferences::MEMORY`.

- the memory database is not shared among multiple instances of the same app
- memory databases are not thread safe
- all data will be lost when the app quits

```cpp
void some_function() {
    Fl_Preferences guide( NULL, "Guide" );
    guide.set("answer", 42);
}
void other_function() {
    int x;
    Fl_Preferences guide( NULL, "Guide" );
    guide.get("answer", x, -1);
}
```

FLTK uses the memory database to manage plugins. See `Fl_Plugin`.

Parameters

| in parent |
| the parameter parent is a pointer to the parent group. If parent is NULL, the new preferences item refers to an application internal database ("runtime prefs") which exists only once, and remains in RAM only until the application quits. This database is used to manage plugins and other data indexes by strings. Runtime prefs are not thread-safe. |
| in group |
| a group name that is used as a key into the database |

See also

`. Fl_Preferences( Fl_Preferences&, const char *group )`

12.105.5 Fl_Preferences() [5/8]

FL_PREFERENCES:

| Fl_Preferences & parent, |
| int groupIndex |

Open a child group using a given index. Use the `groupIndex` argument to find the group that you would like to access. If the given index is invalid (negative or too high), a new group is created with a UUID as a name.
The index needs to be fixed. It is currently backward. Index 0 points to the last member in the 'list' of preferences.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>parent</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>groupIndex</strong></td>
</tr>
</tbody>
</table>

### 12.105.4.6 Fl_Preferences() [6/8]

```c
Fl_Preferences::Fl_Preferences ( 
    Fl_Preferences * parent, 
    int groupIndex )
```

See also

```c
Fl_Preferences( Fl_Preferences&, int groupIndex )
```

### 12.105.4.7 Fl_Preferences() [7/8]

```c
Fl_Preferences::Fl_Preferences ( Fl_Preferences::ID id )
```

Create a new dataset access point using a dataset ID. ID's are a great way to remember shortcuts to database entries that are deeply nested in a preferences database, as long as the database root is not deleted. An ID can be retrieved from any Fl_Preferences dataset, and can then be used to create multiple new references to the same dataset. ID's can be very helpful when put into the user_data() field of widget callbacks.

### 12.105.4.8 ~Fl_Preferences()

```c
Fl_Preferences::~Fl_Preferences ( ) [virtual]
```

The destructor removes allocated resources. When used on the base preferences group, the destructor flushes all changes to the preference file and deletes all internal databases. The destructor does not remove any data from the database. It merely deletes your reference to the database.

### 12.105.4.9 Fl_Preferences() [8/8]

```c
Fl_Preferences::Fl_Preferences ( 
    const char * path, 
    const char * vendor, 
    const char * application )
```

Deprecated: Use this constructor to create or read a preference file at an arbitrary position in the file system.

**Deprecated** "in 1.4.0 - use Fl_Preferences(path, vendor, application, flags) instead"

This constructor should no longer be used because the generated database uses the current locale, making it impossible to exchange floating point settings between machines with different language settings. Use Fl_Preferences(path, vendor, application, Fl_Preferences::C_LOCALE) in new projects and Fl_Preferences(path, vendor, application, 0) if you must keep backward compatibility.

See also

```c
Fl_Preferences( const char *path, const char *vendor, const char *application, Root flags )
```
12.105.5 Member Function Documentation

12.105.5.1 delete_entry()

```
char Fl_Preferences::delete_entry (const char * key )
```

Deletes a single name/value pair.
This function removes the entry `key` from the database.

**Parameters**

- `key` name of entry to delete

**Returns**

- `0` if deleting the entry failed

12.105.5.2 delete_group()

```
char Fl_Preferences::delete_group (const char * group )
```

Deletes a group.
Removes a group and all keys and groups within that group from the database.

**Parameters**

- `group` name of the group to delete

**Returns**

- `0` if call failed

12.105.5.3 dirty()

```
int Fl_Preferences::dirty ( )
```

Check if there were changes to the database that need to be written to disk.

**Returns**

- `1` if the database will be written to disk by `flush` or destructor.
- `0` if the database is unchanged since the last write operation.
- `-1` if there is an internal database error.

12.105.5.4 entries()

```
int Fl_Preferences::entries ( )
```

Returns the number of entries (name/value pairs) in a group.

**Returns**

- number of entries

12.105.5.5 entry()

```
const char * Fl_Preferences::entry (int index )
```

Returns the name of an entry.
There is no guaranteed order of entry names. The index must be within the range given by `entries()`.
### 12.105.5.6 entry_exists()

```cpp
cchar Fl_Preferences::entry_exists (const char * key)
```

Retruns non-zero if an entry with this name exists.

**Parameters**

- **in key** name of entry that is searched for

**Returns**

- 0 if entry was not found

### 12.105.5.7 file_access() [1/2]

```cpp
unsigned int Fl_Preferences::file_access ( ) [static]
```

Return the current file access permissions for the FLTK preferences system.

See also

- Fl_Preferences::file_access(unsigned int)

### 12.105.5.8 file_access() [2/2]

```cpp
void Fl_Preferences::file_access ( unsigned int flags ) [static]
```

Tell the FLTK preferences system which files in the file system it may read, create, or write.

The FLTK core library will try to read or even create or write preference files when calling Fl::option(), Fl_File_Chooser, the printing panel, and possibly some other internal functions. If your application wants to keep FLTK from touching the file system, call this function before making any other FLTK calls:

```cpp
// neither FLTK nor the app may read, create, or write preference files
Fl_Preferences::file_access( Fl_Preferences::NONE);
```

Or

```cpp
// FLTK may not read, create, or write preference files, but the application may
Fl_Preferences::file_access( Fl_Preferences::APP_OK);
```

All flags can be combined using an OR operator. If flags are not set, that specific access to the file system will not be allowed. By default, all access is granted. To clear one or more flags from the default setting, use:

```cpp
Fl_Preferences::file_access( Fl_Preferences::file_access() &~ Fl_Preferences::SYSTEM_WRITE);
```

If preferences are created using a filename (instead of Fl_Preferences::USER or Fl_Preferences::SYSTEM), file access is handled as if the Fl_Preferences::USER flag was set.

See also

- Fl_Preferences::NONE and others for a list of flags.
- Fl_Preferences::file_access()
12.105.5.9  filename() [1/2]

```
Fl_Preferences::Root Fl_Preferences::filename {
    char * buffer,
    size_t buffer_size
}
```

Return the file name and path to the preference file.
If the preferences have not changed or have not been flushed, the file or directory may not have been created yet.

**Parameters**

| out buffer | write the resulting path into this buffer |
| in buffer_size | size of the buffer in bytes |

**Returns**

the root type at creation type, or MEMORY for runtime prefs, it does not return CORE or LOCALE flags.

12.105.5.10  filename() [2/2]

```
Fl_Preferences::Root Fl_Preferences::filename {
    char * buffer,
    size_t buffer_size,
    Root root,
    const char * vendor,
    const char * application } [static]
```

Determine the file name and path to preferences that would be openend with these parameters.
Find the possible location of a preference file on disk without touching any of the pathname components. This can be used to check if a preference file already exists.

**Parameters**

| out buffer | write the resulting path into this buffer |
| in buffer_size | size of the buffer in bytes |
| in root | can be USER_L or SYSTEM_L for user specific or system wide preferences |
| in vendor | unique text describing the company or author of this file, must be a valid filepath segment |
| in application | unique text describing the application, must be a valid filepath segment |

**Returns**

the input root value, or Fl_Preferences::UNKNOWN_ROOT_TYPE if the path could not be determined.

**See also**

Fl_Preferences( Root root, const char *vendor, const char *application )

12.105.5.11  flush()

```
int Fl_Preferences::flush ( )
```

Writes preferences to disk if they were modified.
This method can be used to verify that writing a preference file went well. Deleting the base preferences object will also write the contents of the database to disk.

**Returns**

-1 if anything went wrong, i.e. file could not be opened, permissions blocked writing, etc.
0 if the file was written to disk. This does not check if the disk ran out of space and the file is truncated.
1 if no data was written to the database and no write attempt to disk was made.
12.105.5.12  get() [1/8]

char Fl_Preferences::get (  
  const char * key,  
  char * & text,  
  const char * defaultValue )

Reads an entry from the group.  
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).  
get() allocates memory of sufficient size to hold the value. The buffer must be free'd by the developer using 'free(value)'.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>text</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

12.105.5.13  get() [2/8]

char Fl_Preferences::get (  
  const char * key,  
  char * text,  
  const char * defaultValue,  
  int maxSize )

Reads an entry from the group.  
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).  
'maxSize' is the maximum length of text that will be read. The text buffer must allow for one additional byte for a trailing zero.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>text</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
<tr>
<td>in</td>
<td>maxSize</td>
<td>maximum length of value plus one byte for a trailing zero</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

12.105.5.14  get() [3/8]

char Fl_Preferences::get (  
  const char * key,  
  double & value,  
  double defaultValue )

Reads an entry from the group.  
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>value</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

12.105.15  get() [4/8]

```cpp
char Fl_Preferences::get (const char * key, float & value, float defaultValue )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>value</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

12.105.16  get() [5/8]

```cpp
char Fl_Preferences::get (const char * key, int & value, int defaultValue )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>value</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

12.105.17  get() [6/8]

```cpp
char Fl_Preferences::get (const char * key, 
```
void *data,
const void *defaultValue,
int defaultSize)

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was
used (0). get() allocates memory of sufficient size to hold the value. The buffer must be free’d by the developer
using 'free(value)'.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>data</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultSize</td>
<td>size of default value array</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

12.105.5.18 get() [7/8]

char Fl_Preferences::get (  
    const char * key,
    void * data,
    const void * defaultValue,
    int defaultSize,
    int * maxSize )

Reads a binary entry from the group, encoded in hexadecimal blocks.
A binary (not hex) default value can be supplied. The return value indicates if the value was available (non-zero) or
the default was used (0). maxSize is the maximum length of text that will be read and returns the actual number
of bytes read.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>data</td>
<td>value returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultSize</td>
<td>size of default value array</td>
</tr>
<tr>
<td>in,out</td>
<td>maxSize</td>
<td>maximum length of value and actual number of bytes set</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

12.105.5.19 get() [8/8]

char Fl_Preferences::get (  
    const char * key,
    void * data,
    const void * defaultValue,
    int defaultSize,
    int maxSize )

Reads a binary entry from the group, encoded in hexadecimal blocks.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>data</td>
<td>value returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value</td>
</tr>
<tr>
<td>in</td>
<td>defaultSize</td>
<td>size of default value array</td>
</tr>
<tr>
<td>in</td>
<td>maxSize</td>
<td>maximum length of value, to receive the number of bytes read, use the function below instead.</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

See also

Fl_Preferences::get(const char *key, void *data, const void *defaultValue, int defaultSize, int *maxSize)

12.105.5.20 get_userdata_path()

char Fl_Preferences::get_userdata_path (char *path, int pathlen)

Creates a path that is related to the preference file and that is usable for additional application data. This function creates a directory that is named after the preferences database without the .prefs extension and located in the same directory. It then fills the given buffer with the complete path name.

There is no way to verify that the path name fit into the buffer. If the name is too long, it will be clipped. This function can be used with direct paths that don't end in .prefs. getUserDataPath() will remove any extension and end the path with a /. If the file name has no extension, getUserDataPath() will append .data/ to the path name.

Example:

Fl_Preferences prefs( USER, "matthiasm.com", "test" );
char path[FL_PATH_MAX];
prefs.getUserdataPath( path, FL_PATH_MAX );

creates the preferences database in the directory (User 'matt' on Linux):

/Users/matt/.fltk/matthiasm.com/test.prefs

...and returns the userdata path:

/Users/matt/.fltk/matthiasm.com/test/

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>path</th>
<th>buffer for user data path</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pathlen</td>
<td>size of path buffer (should be at least FL_PATH_MAX)</td>
</tr>
</tbody>
</table>

Returns

1 if there is no filename (path will be unmodified)
1 if pathlen is 0 (path will be unmodified)
1 if a path was created successfully, path will contain the path name ending in a '/'
0 if path was not created for some reason; path will contain the pathname that could not be created

See also

Fl_Preferences::Fl_Preferences(Root, const char*, const char*)

12.105.5.21 group()

const char * Fl_Preferences::group (int num_group)
Returns the name of the Nth \texttt{(num\_group)} group.
There is no guaranteed order of group names. The index must be within the range given by \texttt{groups()}.

**Parameters**

| in | num\_group | number indexing the requested group |

**Returns**

'C' string pointer to the group name

\subsection{group\_exists()}

\begin{verbatim}
char Fl_Preferences::group_exists (const char * key )
\end{verbatim}

Returns non-zero if a group with this name exists.
Group names are relative to the Fl\_Preferences node and can contain a path. '.' describes the current node, '.../' describes the topmost node. By preceding a group name with a '.../' its path becomes relative to the topmost node.

**Parameters**

| in | key | name of group that is searched for |

**Returns**

0 if no group by that name was found

\subsection{groups()}

\begin{verbatim}
int Fl_Preferences::groups ( )
\end{verbatim}

Returns the number of groups that are contained within a group.

**Returns**

0 for no groups at all

\subsection{new\_UUID()}

\begin{verbatim}
const char * Fl_Preferences::new\_UUID ( ) [static]
\end{verbatim}

Returns a UUID as generated by the system.
A UUID is a "universally unique identifier" which is commonly used in configuration files to create identities. A UUID in ASCII looks like this: 937C4900-51AA-4C11-8DD3-7AB59944F03E. It has always 36 bytes plus a trailing zero.

**Returns**

a pointer to a static buffer containing the new UUID in ASCII format. The buffer is overwritten during every call to this function!

\subsection{set() [1/7]}

\begin{verbatim}
char Fl_Preferences::set (const char * key,
                        const char * text )
\end{verbatim}

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>text</td>
<td>set this entry to value</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed

12.105.5.26  set() [2/7]

```
cchar Fl_Preferences::set (  
    const char * key,  
    const void * data,  
    int dsize )  
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>set this entry to value</td>
</tr>
<tr>
<td>in</td>
<td>dsize</td>
<td>size of data array</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed

12.105.5.27  set() [3/7]

```
cchar Fl_Preferences::set (  
    const char * key,  
    double value )  
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>set this entry to value</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed

12.105.5.28  set() [4/7]

```
cchar Fl_PREFERENCES::set (  
    const char * key,  
    double value,  
    int precision )  
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>set this entry to value</td>
</tr>
<tr>
<td>in</td>
<td>precision</td>
<td>number of decimal digits to represent value</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed

12.105.5.29 set() [5/7]

```cpp
char Fl_Preferences::set (const char * key, float value )
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>set this entry to value</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed

12.105.5.30 set() [6/7]

```cpp
char Fl_Preferences::set (const char * key, float value, int precision )
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>set this entry to value</td>
</tr>
<tr>
<td>in</td>
<td>precision</td>
<td>number of decimal digits to represent value</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed

12.105.5.31 set() [7/7]

```cpp
char Fl_Preferences::set (const char * key, int value )
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>set this entry to value</td>
</tr>
</tbody>
</table>

**Returns**

0 if setting the value failed

### 12.105.32 size()

```c++
int Fl_Preferences::size (const char * key )
```

Returns the size of the value part of an entry.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
</table>

**Returns**

size of value

### 12.105.6 Member Data Documentation

#### 12.105.6.1 CORE_READ_OK

```c++
const unsigned int Fl_Preferences::CORE_READ_OK = 0x0010 [static]
```

Set this if it is OK for FLTK to read preference files. USER_READ_OK and/or SYSTEM_READ_OK must also be set.

#### 12.105.6.2 CORE_WRITE_OK

```c++
const unsigned int Fl_Preferences::CORE_WRITE_OK = 0x0020 [static]
```

Set this if it is OK for FLTK to create or write preference files. USER_WRITE_OK and/or SYSTEM_WRITE_OK must also be set.

#### 12.105.6.3 NONE

```c++
const unsigned int Fl_Preferences::NONE = 0x0000 [static]
```

Set this if no call to Fl_Preferences shall access the file system.

See also

- Fl_Preferences::file_access(unsigned int)
- Fl_Preferences::file_access()

The documentation for this class was generated from the following files:

- Fl_Preferences.H
- Fl_Preferences.cxx

### 12.106 Fl_Printer Class Reference

OS-independent print support.
#include <Fl_Printer.H>

Inheritance diagram for Fl_Printer:

```
Fl_Surface_Device
    Fl_Paged_Device
        Fl_Widget_Surface
            Fl_Printer
```

## Public Member Functions

- `int begin_job (int pagecount=0, int *frompage=NULL, int *topage=NULL, char **perr_message=NULL)` FL_OVERRIDE
  
  Begins a print job.

- `int begin_page (void)` FL_OVERRIDE
  
  Begins a new printed page.

- `void end_job (void)` FL_OVERRIDE
  
  To be called at the end of a print job.

- `int end_page (void)` FL_OVERRIDE
  
  To be called at the end of each page.

- `Fl_Printer (void)`
  
  The constructor.

- `bool is_current ()` FL_OVERRIDE
  
  Is this surface the current drawing surface?

- `void margins (int *left, int *top, int *right, int *bottom)` FL_OVERRIDE
  
  Computes the dimensions of margins that lie between the printable page area and the full page.

- `void origin (int *x, int *y)` FL_OVERRIDE
  
  Computes the coordinates of the current origin of graphics functions.

- `void origin (int x, int y)` FL_OVERRIDE
  
  Sets the position of the origin of graphics in the drawable part of the drawing surface.

- `int printable_rect (int *w, int *h)` FL_OVERRIDE
  
  Computes the width and height of the drawable area of the drawing surface.

- `void rotate (float angle)` FL_OVERRIDE
  
  Rotates the graphics operations relatively to paper.

- `void scale (float scale_x, float scale_y=0.)` FL_OVERRIDE
  
  Changes the scaling of page coordinates.

- `void set_current (void)` FL_OVERRIDE
  
  Make this surface the current drawing surface.

- `void translate (int x, int y)` FL_OVERRIDE
  
  Translates the current graphics origin accounting for the current rotation.

- `void untranslate (void)` FL_OVERRIDE
  
  Undoes the effect of a previous translate() call.

- `~Fl_Printer (void)`
  
  The destructor.
Public Member Functions inherited from Fl_Paged_Device

- void print_widget (Fl_Widget *widget, int delta_x=0, int delta_y=0)
  
  Synonym of draw(Fl_Widget*, int, int)

- void print_window (Fl_Window *win, int x_offset=0, int y_offset=0)
  
  Synonym of draw_decorated_window(Fl_Window*, int, int)

- int start_job (int pagecount=0, int *frompage=NULL, int *topage=NULL, char **perr_message=NULL)
  
  Synonym of begin_job(int pagecount, int *frompage, int *topage, char **perr_message).

- int start_page ()
  
  Synonym of begin_page().

- virtual ~Fl_Paged_Device ()
  
  The destructor.

Public Member Functions inherited from Fl_Widget_Surface

- void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)
  
  Draws the widget on the drawing surface.

- void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)
  
  Draws a window with its title bar and frame if any.

- void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)
  
  Draws a rectangular part of an on-screen window.

Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver * driver ()
  
  Returns the graphics driver of this drawing surface.

- virtual ~Fl_Surface_Device ()
  
  The destructor.

Static Public Attributes

These attributes are useful for the Linux/Unix platform only.

- static const char * dialog_title = "Print"
  
  [this text may be customized at run-time]

- static const char * dialog_printer = "Printer:"
  
  [this text may be customized at run-time]

- static const char * dialog_range = "Print Range"
  
  [this text may be customized at run-time]

- static const char * dialog_copies = "Copies"
  
  [this text may be customized at run-time]

- static const char * dialog_all = "All"
  
  [this text may be customized at run-time]

- static const char * dialog_pages = "Pages"
  
  [this text may be customized at run-time]

- static const char * dialog_from = "From:"
  
  [this text may be customized at run-time]

- static const char * dialog_to = "To:"
  
  [this text may be customized at run-time]

- static const char * dialog_properties = "Properties..."
  
  [this text may be customized at run-time]

- static const char * dialog_copyNo = "# Copies:"
  
  [this text may be customized at run-time]

- static const char * dialog_print_button = "Print"
  
  [this text may be customized at run-time]

- static const char * dialog_cancel_button = "Cancel"
• static const char ∗ dialog_print_to_file = "Print To File"
  [this text may be customized at run-time]
• static const char ∗ property_title = "Printer Properties"
  [this text may be customized at run-time]
• static const char ∗ property_pagesize = "Page Size:"
  [this text may be customized at run-time]
• static const char ∗ property_mode = "Output Mode:"
  [this text may be customized at run-time]
• static const char ∗ property_use = "Use"
  [this text may be customized at run-time]
• static const char ∗ property_save = "Save"
  [this text may be customized at run-time]
• static const char ∗ property_cancel = "Cancel"
  [this text may be customized at run-time]

Static Public Attributes inherited from Fl_Paged_Device

• static const page_format page_formats [NO_PAGE_FORMATS]
  width, height and name of all elements of the enum Page_Format.

Additional Inherited Members

Public Types inherited from Fl_Paged_Device

• enum Page_Format {
   A0 = 0 , A1 , A2 , A3 ,
   A4 , A5 , A6 , A7 ,
   A8 , A9 , B0 , B1 ,
   B2 , B3 , B4 , B5 ,
   B6 , B7 , B8 , B9 ,
   B10 , C5E , DLE , EXECUTIVE ,
   FOLIO , LEDGER , LEGAL , LETTER ,
   TABLOID , ENVELOPE , MEDIA = 0x1000 }
   Possible page formats.
• enum Page_Layout { PORTRAIT = 0 , LANDSCAPE = 0x100 , REVERSED = 0x200 , ORIENTATION = 0x300 }
   Possible page layouts.

Static Public Member Functions inherited from Fl_Surface_Device

• static Fl_Surface_Device ∗ pop_current ()
  Removes the top element from the current drawing surface stack, and makes
  the new top element current.
• static void push_current (Fl_Surface_Device ∗ new_current)
  Pushes new_current on top of the stack of current drawing surfaces, and makes it current.
• static Fl_Surface_Device ∗ surface ()
  The current drawing surface.

Protected Member Functions inherited from Fl_Paged_Device

• Fl_Paged_Device ()
  The constructor.

Protected Member Functions inherited from Fl_Widget_Surface

• Fl_Widget_Surface (Fl_Graphics_Driver ∗d)
  The constructor.
Protected Member Functions inherited from Fl_Surface_Device

- void driver (Fl_Graphics_Driver *graphics_driver)
  
  Sets the graphics driver of this drawing surface.

- virtual void end_current ()
  
  FLTK calls this each time a surface ceases to be the current drawing surface.

- Fl_Surface_Device (Fl_Graphics_Driver *graphics_driver)
  
  Constructor that sets the graphics driver to use for the created surface.

Protected Attributes inherited from Fl_Widget_Surface

- int x_offset
  
  horizontal offset to the origin of graphics coordinates

- int y_offset
  
  vertical offset to the origin of graphics coordinates

12.106.1 Detailed Description

OS-independent print support.

Fl_Printer allows to use all drawing, color, text, image, and clip FLTK functions, and to have them operate on printed page(s). There are two main, non exclusive, ways to use it.

- Print any widget (standard, custom, Fl_Window, Fl_Gl_Window) as it appears on screen, with optional translation, scaling and rotation. This is done by calling print_widget(), print_window() or print_window_part().

- Use a series of FLTK graphics commands (e.g., font, text, lines, colors, clip, image) to compose a page appropriately shaped for printing.

In both cases, begin by begin_job(), begin_page(), printable_rect() and origin() calls and finish by end_page() and end_job() calls.

Example of use: print a widget centered in a page

```c
#include <FL/Fl_Printer.H>
#include <FL/fl_draw.H>

int width, height;
Fl_Widget *widget = ... // a widget we want printed
Fl_Printer *printer = new Fl_Printer();
if (printer->begin_job(1) == 0) {
    printer->begin_page();
    printer->printable_rect(&width, &height);
    fl_color(FL_BLACK);
    fl_line_style(FL_SOLID, 2);
    fl_rect(0, 0, width, height);
    fl_font(FL_COURIER, 12);
    time_t now; time(&now); fl_draw(ctime(&now), 0, fl_height());
    printer->origin(width/2, height/2);
    printer->print_widget(widget, -widget->w()/2, -widget->h()/2);
    printer->end_page();
    printer->end_job();
}
delete printer;
```

Recommended method to refresh GUI while printing:

```c
printer->begin_job(0);

......
Fl_Surface_Device::push_current(Fl_Display_Device::display_device());
Fl::check(); // or any operation that draws to display
Fl_Surface_Device::pop_current();

......
printer->end_job();
```

Platform specifics

- X11 and Wayland platforms:
  - FLTK expresses all graphics data using (Level 2) PostScript and sends that to the selected printer. See class Fl_PostScript_File_Device for a description of how text and transparent images appear in print.
  - If the GTK library is available at run-time, class Fl_Printer runs GTK’s printer dialog which allows to set printer, paper size and orientation.
– Under the KDE desktop, Fl_Printer runs the kdialog command to create KDE-styled file dialogs if that command is available at run-time, unless FLTK was built with CMake and option OPTION_USE_KDIALOG turned off. In that case, Fl_Printer attempts to run the GTK dialog.

– If the GTK library is not available, or if Fl::option(Fl::OPTION_PRINTER_USES_GTK) has been turned off, class Fl_Printer runs FLTK’s print dialog.

• Unless it has been previously changed, the default paper size is A4. To change that, press the "Properties" button of the "Print" dialog window opened by an Fl_Printer::begin_job() call. This opens a "Printer Properties" window where it’s possible to select the adequate paper size. Finally press the "Save" button therein to assign the chosen paper size to the chosen printer for this and all further print operations.

• Use the static public attributes of this class to set the print dialog to other languages than English. For example, the “Printer:” dialog item Fl_Printer::dialog_printer can be set to French with:
  Fl_Printer::dialog_printer = "Imprimante:"

  before creation of the Fl_Printer object.

• Use Fl_PostScript_File_Device::file_chooser_title to customize the title of the file chooser dialog that opens when using the "Print To File" option of the print dialog.

• Windows platform: Transparent Fl_RGB_Images don’t print with exact transparency on most printers (a workaround is to use print_window_part()). Fl_RGB_Images don’t rotate() well.

• Mac OS X platform: all graphics requests print as on display and accept rotation and scaling.

12.106.2 Member Function Documentation

12.106.2.1 begin_job()

```cpp
int Fl_Printer::begin_job ( 
    int pagecount = 0, 
    int * frompage = NULL, 
    int * topage = NULL, 
    char ** perr_message = NULL ) [virtual]
```

Begins a print job.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pagecount</td>
<td>the total number of pages of the job (or 0 if you don’t know the number of pages)</td>
</tr>
<tr>
<td>frompage</td>
<td>if non-null, +frompage is set to the first page the user wants printed</td>
</tr>
<tr>
<td>topage</td>
<td>if non-null, +topage is set to the last page the user wants printed</td>
</tr>
<tr>
<td>perr_message</td>
<td>if non-null and if the returned value is 2, +perr_message is set to a string describing the error. That string can be delete[]d after use.</td>
</tr>
</tbody>
</table>

Returns

0 if OK, 1 if user cancelled the job, 2 if any error.

Reimplemented from Fl_Paged_Device.

12.106.2.2 begin_page()

```cpp
int Fl_Printer::begin_page ( 
    void ) [virtual]
```

Begins a new printed page.

The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area. This function also makes this surface the current drawing surface with Fl_Surface_Device::push_current().
begin_page() calls Fl_Surface_Device::push_current() and leaves this device as the active surface. If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current(), or the content of the dialog box will be rendered to the printer instead of the screen.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Paged_Device.

12.106.2.3 end_job()

void Fl_Printer::end_job (
    void ) [virtual]

To be called at the end of a print job.
Reimplemented from Fl_Paged_Device.

12.106.2.4 end_page()

int Fl_Printer::end_page (
    void ) [virtual]

To be called at the end of each page.
This function also stops this surface from being the current drawing surface with Fl_Surface_Device::pop_current().

Note

end_page() calls Fl_Surface_Device::pop_current(). If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current().

Returns

0 if OK, non-zero if any error.

Reimplemented from Fl_Paged_Device.

12.106.2.5 is_current()

bool Fl_Printer::is_current ( ) [virtual]

Is this surface the current drawing surface?
Reimplemented from Fl_Surface_Device.

12.106.2.6 margins()

void Fl_Printer::margins (  
    int * left,    
    int * top,    
    int * right,  
    int * bottom ) [virtual]

Computes the dimensions of margins that lie between the printable page area and the full page. Values are in the same unit as that used by FLTK drawing functions. They are changed by scale() calls.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>If non-null, *left is set to the left margin size.</td>
</tr>
<tr>
<td>top</td>
<td>If non-null, *top is set to the top margin size.</td>
</tr>
<tr>
<td>right</td>
<td>If non-null, *right is set to the right margin size.</td>
</tr>
<tr>
<td>bottom</td>
<td>If non-null, *bottom is set to the bottom margin size.</td>
</tr>
</tbody>
</table>
origin() [1/2]

```cpp
void Fl_Printer::origin (  
    int * x,  
    int * y ) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

**Parameters**
- `out x,y` If non-null, `*x` and `*y` are set to the horizontal and vertical coordinates of the graphics origin.

origin() [2/2]

```cpp
void Fl_Printer::origin (  
    int x,  
    int y ) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.

Arguments should be expressed relatively to the result of a previous `printable_rect()` call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive `origin()` calls don't combine their effects. `Origin()` calls are not affected by `rotate()` calls (for classes derived from `Fl_Paged_Device`).

**Parameters**
- `in x,y` Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

printable_rect()

```cpp
int Fl_Printer::printable_rect (  
    int * w,  
    int * h ) [virtual]
```

Computes the width and height of the drawable area of the drawing surface.

Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to `origin()`. If the object is derived from class `Fl_Paged_Device`, values account for the user-selected paper type and print orientation and are changed by `scale()` calls.

**Returns**
- 0 if OK, non-zero if any error

rotate()

```cpp
void Fl_Printer::rotate (  
    float angle ) [virtual]
```

Rotates the graphics operations relatively to paper.

The rotation is centered on the current graphics origin. Successive `rotate()` calls don't combine their effects.

**Parameters**
- `angle` Rotation angle in counter-clockwise degrees.
The documentation for this class was generated from the following files:

Reimplemented from `Fl_Paged_Device`.

### 12.106.2.11 scale()

```cpp
void Fl_Printer::scale ( 
    float scale_x, 
    float scale_y = 0. ) [virtual]
```

Changes the scaling of page coordinates. This function also resets the origin of graphics functions at top left of printable page area. After a `scale()` call, do a `printable_rect()` call to get the new dimensions of the printable page area. Successive `scale()` calls don’t combine their effects.

#### Parameters

| scale← _x | Horizontal dimensions of plot are multiplied by this quantity. |
| scale← _y | Same as above, vertically. The value 0. is equivalent to setting scale_y = scale_x. Thus, scale(factor); is equivalent to scale(factor, factor); |

Reimplemented from `Fl_Paged_Device`.

### 12.106.2.12 set_current()

```cpp
void Fl_Printer::set_current ( 
    void ) [virtual]
```

Make this surface the current drawing surface. This surface will receive all future graphics requests. Starting from FLTK 1.4.0, the preferred API to change the current drawing surface is `Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current()`.

#### Note

It’s recommended to use this function only as follows:

- The current drawing surface is the display;
- make current another surface, e.g., an `Fl_Printer` or an `Fl_Image_Surface` object, calling `set_current()` on this object;
- draw to that surface;
- make the display current again with `Fl_Display_Device::display_device()->set_current();`. Don’t do any other call to `set_current()` before this one.

Other scenarios of drawing surface changes should be performed via `Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current()`.

Reimplemented from `Fl_Surface_Device`.

### 12.106.2.13 translate()

```cpp
void Fl_Printer::translate ( 
    int x, 
    int y ) [virtual]
```

Translates the current graphics origin accounting for the current rotation. Each `translate()` call must be matched by an `untranslate()` call. Successive `translate()` calls add up their effects.

Reimplemented from `Fl_Widget_Surface`.

### 12.106.2.14 untranslate()

```cpp
void Fl_Printer::untranslate ( 
    void ) [virtual]
```

Undoes the effect of a previous `translate()` call.

Reimplemented from `Fl_Widget_Surface`.

The documentation for this class was generated from the following files:
Fl_Progress Class Reference

Displays a progress bar for the user.

```c
#include <Fl_Progress.H>
```

Inheritance diagram for Fl_Progress:

```
Fl_Widget

| Fl_Progress
```

Public Member Functions

- **Fl_Progress** (int x, int y, int w, int h, const char *l=0)
  
  The constructor creates the progress bar using the position, size, and label.

- float maximum () const
  
  Gets the maximum value in the progress widget.

- void maximum (float v)
  
  Sets the maximum value in the progress widget.

- float minimum () const
  
  Gets the minimum value in the progress widget.

- void minimum (float v)
  
  Sets the minimum value in the progress widget.

- float value () const
  
  Gets the current value in the progress widget.

- void value (float v)
  
  Sets the current value in the progress widget.

Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  
  Activates the widget.

- unsigned int active () const
  
  Returns whether the widget is active.

- int active_r () const
  
  Returns whether the widget and all of its parents are active.

- Fl_Align align () const
  
  Gets the label alignment.

- void align (Fl_Align alignment)
  
  Sets the label alignment.

- long argument () const
  
  Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)
  
  Sets the current user data (long) argument that is passed to the callback function.

- virtual class Fl_Gl_Window * as_gl_window ()
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const * as_gl_window () const
• virtual Fl_Group * as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• virtual Fl_Group const * as_group () const
• virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const * as_window () const
• void bind_deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• Fl_Boxtype box () const
  Gets the box type of the widget.
• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
• Fl_Callback_p callback () const
  Gets the current callback function for the widget.
• void callback (Fl_Callback *cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
  Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• `Fl_Color color2 () const`
  For back compatibility only.
• `void color2 (unsigned a)`
  For back compatibility only.
• `int contains (const Fl_Widget *w) const`
  Checks if `w` is a child of this widget.
• `void copy_label (const char *new_label)`
  Sets the current label.
• `void copy_tooltip (const char *text)`
  Sets the current tooltip text.
• `uchar damage () const`
  Returns non-zero if `draw()` needs to be called.
• `void damage (uchar c)`
  Sets the damage bits for the widget.
• `void damage (uchar c, int x, int y, int w, int h)`
  Sets the damage bits for an area inside the widget.
• `int damage_resize (int, int, int, int)`
  Internal use only.
• `void deactivate ()`
  Deactivates the widget.
• `Fl_Image *deimage ()`
  Gets the image that is used as part of the widget label when in the inactive state.
• `const Fl_Image *deimage () const`
  Gets the image that is used as part of the widget label when in the inactive state.
• `void deimage (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the inactive state.
• `void deimage (Fl_Image *img)`
• `int deimage_bound () const`
  Returns whether the inactive image is managed by the widget.
• `void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with default arguments.
• `void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.
• `void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.
• `void draw_label (int, int, int, int, Fl_Align) const`
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• `int h () const`
  Gets the widget height.
• `virtual int handle (int event)`
  Handles the specified event.
• `virtual void hide ()`
  Makes a widget invisible.
• `Fl_Image *image ()`
  Gets the image that is used as part of the widget label when in the active state.
• `const Fl_Image *image () const`
  Gets the image that is used as part of the widget label when in the active state.
• `void image (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the active state.
• `void image (Fl_Image *img)`
Sets the image to use as part of the widget label when in the active state.

- `int image_bound() const`
  Returns whether the image is managed by the widget.

- `int inside(const Fl_Widget *wgt) const`
  Checks if this widget is a child of `wgt`.

- `int is_label_copied() const`
  Returns whether the current label was assigned with `copy_label()`.

- `const char * label() const`
  Gets the current label text.

- `void label(const char *text)`
  Sets the current label pointer.

- `void label(Fl_Labeltype a, const char *b)`
  Shortcut to set the label text and type in one call.

- `FL_Color labelcolor() const`
  Gets the label color.

- `void labelcolor(FL_Color c)`
  Sets the label color.

- `FL_Font labelfont() const`
  Gets the font to use.

- `void labelfont(FL_Font f)`
  Sets the font to use.

- `FL_Fontsize labelsize() const`
  Gets the font size in pixels.

- `void labelsize(FL_Fontsize pix)`
  Sets the font size in pixels.

- `FL_Labeltype labeltype() const`
  Gets the label type.

- `void labeltype(FL_Labeltype a)`
  Sets the label type.

- `void measure_label(int &ww, int &hh) const`
  Sets width `ww` and height `hh` accordingly with the label size.

- `bool needs_keyboard() const`
  Returns whether this widget needs a keyboard.

- `void needs_keyboard(bool needs)`
  Sets whether this widget needs a keyboard.

- `unsigned int output() const`
  Returns if a widget is used for output only.

- `FL_Group *parent() const`
  Returns a pointer to the parent widget.

- `void parent(FL_Group *p)`
  Internal use only - “for hacks only”.

- `void position(int X, int Y)`
  Repositions the window or widget.

- `void redraw()`
  Schedules the drawing of the widget.

- `void redraw_label()`
  Schedules the drawing of the label.

- `virtual void resize(int x, int y, int w, int h)`
  Changes the size or position of the widget.

- `FL_Color selection_color() const`
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void show ()
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int testShortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.
• const char∗ tooltip () const
  Gets the current tooltip text.
• void tooltip (const char∗ text)
  Sets the current tooltip text.
• Fl_Window∗ top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window∗ top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void ∗ user_data () const
  Gets the user data for this widget.
• void user_data (FI_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void ∗v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
Modifies keyboard focus navigation.

- int `visible_r` () const
  Returns whether a widget and all its parents are visible.

- int `w` () const
  Gets the widget width.

- `FL_When` `when` () const
  Returns the conditions under which the callback is called.

- void `when` (uchar i)
  Sets the flags used to decide when a callback is called.

- `Fl_Window` ∗ `window` () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

- int `x` () const
  Gets the widget position in its window.

- int `y` () const
  Gets the widget position in its window.

- virtual `~Fl_Widget` ()
  Destroys the widget.

Protected Member Functions

- void `draw` () `FL_OVERRIDE`
  Draws the progress bar.

Protected Member Functions inherited from `Fl_Widget`

- void `clear_flag` (unsigned int c)
  Clears a flag in the flags mask.

- void `draw_backdrop` () const
  If `FL_ALIGN_IMAGE_BACKDROP` is set, the image or deimage will be drawn.

- void `draw_box` () const
  Draws the widget box according its box style.

- void `draw_box` (`Fl_Boxtype` t, `Fl_Color` c) const
  Draws a box of type t, of color c at the widget's position and size.

- void `draw_box` (`Fl_Boxtype` t, int x, int y, int w, int h, `Fl_Color` c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- void `draw_focus` () const
  Draws a focus rectangle around the widget.

- void `draw_focus` (`Fl_Boxtype` t, int x, int y, int w, int h, `Fl_Color` bg) const
  Draws a focus box for the widget at the given position and size.

- void `draw_label` () const
  Draws the widget's label at the defined label position.

- void `draw_label` (int, int, int, int) const
  Draws the label in an arbitrary bounding box.

- `Fl_Widget` (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.

- unsigned int `flags` () const
  Gets the widget flags mask.

- void `h` (int v)
  Internal use only.

- void `set_flag` (unsigned int c)
Sets a flag in the flags mask.
• void w (int v)
  Internal use only.
• void x (int v)
  Internal use only.
• void y (int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don’t set a callback.
• static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the ’&x’ shortcut in a given text.
• static int test_shortcut (const char *t, const bool require_alt=false)
  Returns true if the given text t contains the entered ’&x’ shortcut.

Protected Types inherited from Fl_Widget

• enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  , OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  , MENU_WINDOW = 1<<12 , TOOLIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  , GROUP_RELATIVE = 1<<16 , COPYED_TOOLIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
    = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<25 , USERFLAG2 = 1<<26 , USERFLAG1 = 1<<27
  , flags possible values enumeration.
  
12.107.1 Detailed Description

Displays a progress bar for the user.

12.107.2 Constructor & Destructor Documentation

12.107.2.1 Fl_Progress()

Fl_Progress::Fl_Progress ( 
  int X,
  int Y,
  int W,
  int H,
  const char * L = 0 )

The constructor creates the progress bar using the position, size, and label.
You can set the background color with color() and the progress bar color with selection_color(), or you can set both colors together with color(unsigned bg, unsigned sel).
The default colors are FL_BACKGROUND2_COLOR and FL_YELLOW, resp.
12.107.3 Member Function Documentation

12.107.3.1 draw()

```c
void Fl_Progress::draw {
    void ) [protected], [virtual]
```

Draws the progress bar.
Implements Fl_Widget.

12.107.3.2 maximum() [1/2]

```c
float Fl_Progress::maximum ( ) const [inline]
```

Gets the maximum value in the progress widget.

12.107.3.3 maximum() [2/2]

```c
void Fl_Progress::maximum ( float v ) [inline]
```

Sets the maximum value in the progress widget.

12.107.3.4 minimum() [1/2]

```c
float Fl_Progress::minimum ( ) const [inline]
```

Gets the minimum value in the progress widget.

12.107.3.5 minimum() [2/2]

```c
void Fl_Progress::minimum ( float v ) [inline]
```

Sets the minimum value in the progress widget.

12.107.3.6 value() [1/2]

```c
float Fl_Progress::value ( ) const [inline]
```

Gets the current value in the progress widget.

12.107.3.7 value() [2/2]

```c
void Fl_Progress::value ( float v ) [inline]
```

Sets the current value in the progress widget.

The documentation for this class was generated from the following files:

- Fl_Progress.H
- Fl_Progress.hxx
12.108 Fl_Radio_Button Class Reference

Inheritance diagram for Fl_Radio_Button:

```
Fl_Widget
   |
   v
Fl_Button
   |
   v
Fl_Radio_Button
```

Public Member Functions

- **Fl_Radio_Button** (int X, int Y, int W, int H, const char *L=0)
  
  The constructor creates the button using the given position, size, and label.

Public Member Functions inherited from Fl_Button

- int **clear** ()
  
  Same as value(0).
- uchar **compact** ()
  
  Return true if buttons are rendered as compact buttons.
- void **compact** (uchar v)
  
  Decide if buttons should be rendered in compact mode.
- Fl_Boxtype **down_box** () const
  
  Returns the current down box type, which is drawn when value() is non-zero.
- void **down_box** (Fl_Boxtype b)
  
  Sets the down box type.
- Fl_Color **down_color** () const
  
  (for backwards compatibility)
- void **down_color** (unsigned c)
  
  (for backwards compatibility)
- **Fl_Button** (int X, int Y, int W, int H, const char *L=0)
  
  The constructor creates the button using the given position, size, and label.
- int **handle** (int) FL_OVERRIDE
  
  Handles the specified event.
- int **set** ()
  
  Same as value(1).
- void **setonly** ()
  
  Turns on this button and turns off all other radio buttons in the group (calling value(1) or set() does not do this).
- int **shortcut** () const
  
  Returns the current shortcut key for the button.
- void **shortcut** (const char *s)
  
  (for backwards compatibility)
- void **shortcut** (int s)
  
  Sets the shortcut key to s.
- char **value** () const
  
  Returns the current value of the button (0 or 1).
- int **value** (int v)
  
  Sets the current value of the button.
Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  Activates the widget.
• unsigned int active () const
  Returns whether the widget is active.
• int active_r () const
  Returns whether the widget and all of its parents are active.
• Fl_Align align () const
  Gets the label alignment.
• void align (Fl_Align alignment)
  Sets the label alignment.
• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
• virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const ∗ as_gl_window () const
• virtual Fl_Group ∗ as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• virtual Fl_Group const ∗ as_group () const
• virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const ∗ as_window () const
• void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• Fl_Boxtype box () const
  Gets the box type of the widget.
• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
• Fl_Callback_p callback () const
  Gets the current callback function for the widget.
• void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
Checks if the widget value changed since the last callback.

- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  Clears or sets the damage flags.

- void clear_output ()
  Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.

- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

- Fl_Color color () const
  Gets the background color of the widget.

- void color (Fl_Color bg)
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

- Fl_Color color2 () const
  For back compatibility only.

- void color2 (unsigned a)
  For back compatibility only.

- int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.

- void copy_label (const char *new_label)
  Sets the current label.

- void copy_tooltip (const char *text)
  Sets the current tooltip text.

- uchar damage () const
  Returns non-zero if draw() needs to be called.

- void damage (uchar c)
  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- int damage_resize (int, int, int, int)
  Internal use only.

- void deactivate ()
  Deactivates the widget.

- FI_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

- const FI_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

- void deimage (FI_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

- void deimage (FI_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget ∗widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget ∗widget, void ∗arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image ∗image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image ∗image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget ∗wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char ∗label () const
  Gets the current label text.
• void label (const char ∗text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char ∗b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
Sets whether this widget needs a keyboard.

- unsigned int `output () const`
  Returns if a widget is used for output only.

- `Fl_Group * parent () const`
  Returns a pointer to the parent widget.

- void `parent (Fl_Group *p)`
  Internal use only - "for hacks only".

- void `position (int X, int Y)`
  Repositions the window or widget.

- void `redraw ()`
  Schedules the drawing of the widget.

- void `redraw_label ()`
  Schedules the drawing of the label.

- virtual void `resize (int x, int y, int w, int h)`
  Changes the size or position of the widget.

- `Fl_Color selection_color () const`
  Gets the selection color.

- void `selection_color (Fl_Color a)`
  Sets the selection color.

- void `set_active ()`
  Marks the widget as active without sending events or changing focus.

- void `set_changed ()`
  Marks the value of the widget as changed.

- void `set_output ()`
  Sets a widget to output only.

- void `set_visible ()`
  Makes the widget visible.

- void `set_visible_focus ()`
  Enables keyboard focus navigation with this widget.

- int `shortcut_label () const`
  Returns whether the widget's label uses '&' to indicate shortcuts.

- void `shortcut_label (int value)`
  Sets whether the widget's label uses '&' to indicate shortcuts.

- virtual void `show ()`
  Makes a widget visible.

- void `size (int W, int H)`
  Changes the size of the widget.

- int `take_focus ()`
  Gives the widget the keyboard focus.

- unsigned int `takesevents () const`
  Returns if the widget is able to take events.

- int `test_shortcut ()`
  Returns true if the widget's label contains the entered '&x' shortcut.

- const char * `tooltip () const`
  Gets the current tooltip text.

- void `tooltip (const char *text)`
  Sets the current tooltip text.

- `Fl_Window * top_window () const`
  Returns a pointer to the top-level window for the widget.

- `Fl_Window * top_window_offset (int &xoff, int &yoff) const`
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  
  Gets the widget type.

• void type (uchar t)
  
  Sets the widget type.

• int use_accents_menu ()
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void * user_data () const
  
  Gets the user data for this widget.

• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  
  Sets the user data for this widget.

• void user_data (void *v)
  
  Sets the user data for this widget.

• unsigned int visible () const
  
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  
  Checks whether this widget has a visible focus.

• void visible_focus (int v)
  
  Modifies keyboard focus navigation.

• int visible_r () const
  
  Returns whether a widget and all its parents are visible.

• int w () const
  
  Gets the widget width.

• Fl_When when () const
  
  Returns the conditions under which the callback is called.

• void when (uchar i)
  
  Sets the flags used to decide when a callback is called.

• Fl_Window * window () const
  
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  
  Gets the widget position in its window.

• int y () const
  
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  
  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget *widget, void *data)
  
  The default callback for all widgets that don’t set a callback.

• static unsigned int labelShortcut (const char *t)
  
  Returns the Unicode value of the \'&x\' shortcut in a given text.

• static int testShortcut (const char *, const bool require_alt=false)
  
  Returns true if the given text contains the entered \'&x\' shortcut.
Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7
  , OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11
  , MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15
  , GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
    = 1<<19, NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
flags possible values enumeration.

Protected Member Functions inherited from Fl/Button

- void draw () FL_OVERRIDE
  Draws the widget.

- void simulate_key_action ()

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.

- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void draw_box () const
  Draws the widget box according its box style.

- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- void draw_focus () const
  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

- void draw_label () const
  Draws the widget's label at the defined label position.

- void draw_label (int, int, int) const
  Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.

- unsigned int flags () const
  Gets the widget flags mask.

- void h (int v)
  Internal use only.

- void set_flag (unsigned int c)
  Sets a flag in the flags mask.

- void w (int v)
Internal use only.
• void x (int v)
  Internal use only.
• void y (int v)
  Internal use only.

Static Protected Member Functions inherited from Fl_Button
  • static void key_release_timeout (void *)

Static Protected Attributes inherited from Fl_Button
  • static Fl_Widget_Tracker * key_release_tracker = 0

12.108.1 Constructor & Destructor Documentation

12.108.1.1 Fl_Radio_Button()

Fl_Radio_Button::Fl_Radio_Button (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char * L = 0)

The constructor creates the button using the given position, size, and label.  
The Button type() is set to FL_RADIO_BUTTON.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y,W,H</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:
• Fl_Radio_Button.H
• Fl_Button.cxx

12.109 Fl_Radio_Light_Button Class Reference

Inheritance diagram for Fl_Radio_Light_Button:

```
Fl_Widget
    |
    +--------
    |        |
    |        +--------
    |        |        |
    |        |        +--------
    |        |        |        |
    |        |        |        +--------
    |        |        |        |        |
    |        |        |        |        +--------
    |        |        |        |        |        |
    |        |        |        |        |        +--------
    |        |        |        |        |        |        |
    |        |        |        |        |        |        +--------
    |        |        |        |        |        |        |        |
    |        |        |        |        |        |        |        +--------
```

Public Member Functions

• Fl_Radio_Light_Button (int X, int Y, int W, int H, const char *=l=0)
Public Member Functions inherited from Fl_Light_Button

• Fl_Light_Button (int x, int y, int w, int h, const char ∗l=0)
  Creates a new Fl_Light_Button widget using the given position, size, and label string.
• int handle (int) FL_OVERRIDE
  Handles the specified event.

Public Member Functions inherited from Fl_Button

• int clear ()
  Same as value(0).
• uchar compact ()
  Return true if buttons are rendered as compact buttons.
• void compact (uchar v)
  Decide if buttons should be rendered in compact mode.
• Fl_Boxtype down_box () const
  Returns the current down box type, which is drawn when value() is non-zero.
• void down_box (Fl_Boxtype b)
  Sets the down box type.
• Fl_Color down_color () const
  (for backwards compatibility)
• void down_color (unsigned c)
  (for backwards compatibility)
• Fl_Button (int X, int Y , int W, int H, const char ∗L=0)
  The constructor creates the button using the given position, size, and label.
• int set ()
  Same as value(1).
• void setonly ()
  Turns on this button and turns off all other radio buttons in the group. (calling value(1) or set() does not do this).
• int shortcut () const
  Returns the current shortcut key for the button.
• void shortcut (const char ∗s)
  (for backwards compatibility)
• void shortcut (int s)
  Sets the shortcut key to s.
• char value () const
  Returns the current value of the button (0 or 1).
• int value (int v)
  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  Activates the widget.
• unsigned int active () const
  Returns whether the widget is active.
• int active_r () const
  Returns whether the widget and all of its parents are active.
• Fl_Align align () const
  Gets the label alignment.
• void **align**(Fl_Align alignment)
  
  Sets the label alignment.

• long **argument** () const
  
  Gets the current user data (long) argument that is passed to the callback function.

• void **argument**(long v)
  
  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window ∗ **as_gl_window** ()
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const ∗ **as_gl_window** () const

• virtual Fl_Group ∗ **as_group** ()
  
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Group const ∗ **as_group** () const

• virtual Fl_Window ∗ **as_window** ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window const ∗ **as_window** () const

• void **bind_deimage**(Fl_Image ∗ img)
  
  Sets the image to use as part of the widget label when in the inactive state.

• void **bind_deimage**(int f)
  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

• void **bind_image**(Fl_Image ∗ img)
  
  Sets the image to use as part of the widget label when in the active state.

• void **bind_image**(int f)
  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

• Fl_Boxtype **box** () const
  
  Gets the box type of the widget.

• void **box**(Fl_Boxtype new_box)
  
  Sets the box type for the widget.

• Fl_Callback_p **callback** () const
  
  Gets the current callback function for the widget.

• void **callback**(Fl_Callback ∗ cb)
  
  Sets the current callback function for the widget.

• void **callback**(Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  
  Sets the current callback function and managed user data for the widget.

• void **callback**(Fl_Callback ∗ cb, void ∗ p)
  
  Sets the current callback function and data for the widget.

• void **callback**(Fl_Callback0 ∗ cb)
  
  Sets the current callback function for the widget.

• void **callback**(Fl_Callback1 ∗ cb, long p=0)
  
  Sets the current callback function for the widget.

• unsigned int **changed** () const
  
  Checks if the widget value changed since the last callback.

• void **clear_active** ()
  
  Marks the widget as inactive without sending events or changing focus.

• void **clear_changed** ()
  
  Marks the value of the widget as unchanged.

• void **clear_damage**(uchar c=0)
  
  Clears or sets the damage flags.

• void **clear_output** ()
  
  Sets a widget to accept input.

• void **clear_visible** ()
  
  Hides the widget.
• void clear_visible_focus ()
  *Disables keyboard focus navigation with this widget.*

• Fl_Color color () const
  *Gets the background color of the widget.*

• void color (Fl_Color bg)
  *Sets the background color of the widget.*

• void color (Fl_Color bg, Fl_Color sel)
  *Sets the background and selection color of the widget.*

• Fl_Color color2 () const
  *For back compatibility only.*

• void color2 (unsigned a)
  *For back compatibility only.*

• int contains (const Fl_Widget ∗w) const
  *Checks if w is a child of this widget.*

• void copy_label (const char ∗new_label)
  *Sets the current label.*

• void copy_tooltip (const char ∗text)
  *Sets the current tooltip text.*

• uchar damage () const
  *Returns non-zero if draw() needs to be called.*

• void damage (uchar c)
  *Sets the damage bits for the widget.*

• void damage (uchar c, int x, int y, int w, int h)
  *Sets the damage bits for an area inside the widget.*

• int damage_resize (int, int, int, int)
  *Internal use only.*

• void deactivate ()
  *Deactivates the widget.*

• Fl_Image ∗deimage ()
  *Gets the image that is used as part of the widget label when in the inactive state.*

• const Fl_Image ∗deimage () const
  *Gets the image that is used as part of the widget label when in the inactive state.*

• void deimage (Fl_Image &img)
  *Sets the image to use as part of the widget label when in the inactive state.*

• void deimage (Fl_Image ∗img)
  *Sets the image to use as part of the widget label when in the inactive state.*

• int deimage_bound () const
  *Returns whether the inactive image is managed by the widget.*

• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  *Calls the widget callback function with default arguments.*

• void do_callback (Fl_Widget ∗widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  *Calls the widget callback function with arbitrary arguments.*

• void do_callback (Fl_Widget ∗widget, void ∗arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  *Calls the widget callback function with arbitrary arguments.*

• void draw_label (int, int, int, int, Fl_Align) const
  *Draws the label in an arbitrary bounding box with an arbitrary alignment.*

• int h () const
  *Gets the widget height.*

• virtual void hide ()
  *Makes a widget invisible.*

• Fl Image ∗image ()
Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image** ∗ image () const
  
  Gets the image that is used as part of the widget label when in the active state.

- **void image (Fl_Image &img)**
  
  Sets the image to use as part of the widget label when in the active state.

- **void image (Fl_Image ∗img)**
  
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const**
  
  Returns whether the image is managed by the widget.

- **int inside (const Fl_Widget ∗wgt) const**
  
  Checks if this widget is a child of wgt.

- **int is_label_copied () const**
  
  Returns whether the current label was assigned with copy_label().

- **const char ∗ label () const**
  
  Gets the current label text.

- **void label (const char ∗text)**
  
  Sets the current label pointer.

- **void label (Fl_Labeltype a, const char ∗b)**
  
  Shortcut to set the label text and type in one call.

- **Fl_Color labelcolor () const**
  
  Gets the label color.

- **void labelcolor (Fl_Color c)**
  
  Sets the label color.

- **Fl_Font labelfont () const**
  
  Gets the font to use.

- **void labelfont (Fl_Font f)**
  
  Sets the font to use.

- **Fl_Fontsize labelsize () const**
  
  Gets the font size in pixels.

- **void labelsize (Fl_Fontsize pix)**
  
  Sets the font size in pixels.

- **Fl_Labeltype labeltype () const**
  
  Gets the label type.

- **void labeltype (Fl_Labeltype a)**
  
  Sets the label type.

- **void measure_label (int &ww, int &hh) const**
  
  Sets width ww and height hh accordingly with the label size.

- **bool needs_keyboard () const**
  
  Returns whether this widget needs a keyboard.

- **void needs_keyboard (bool needs)**
  
  Sets whether this widget needs a keyboard.

- **unsigned int output () const**
  
  Returns if a widget is used for output only.

- **Fl_Group ∗ parent () const**
  
  Returns a pointer to the parent widget.

- **void parent (Fl_Group ∗p)**
  
  Internal use only - “for hacks only”.

- **void position (int X, int Y)**
  
  Repositions the window or widget.

- **void redraw ()**
  
  Schedules the drawing of the widget.
• void **redraw_label** ()
  Schedules the drawing of the label.
• virtual void **resize** (int x, int y, int w, int h)
  Changes the size or position of the widget.
• **Fl_Color selection_color** () const
  Gets the selection color.
• void **selection_color** (Fl_Color a)
  Sets the selection color.
• void **set_active** ()
  Marks the widget as active without sending events or changing focus.
• void **set_changed** ()
  Marks the value of the widget as changed.
• void **set_output** ()
  Sets a widget to output only.
• void **set_visible** ()
  Makes the widget visible.
• void **set_visible_focus** ()
  Enables keyboard focus navigation with this widget.
• int **shortcut_label** () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void **shortcut_label** (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void **show** ()
  Makes a widget visible.
• void **size** (int W, int H)
  Changes the size of the widget.
• int **take_focus** ()
  Gives the widget the keyboard focus.
• unsigned int **takesevents** () const
  Returns if the widget is able to take events.
• int **test_shortcut** ()
  Returns true if the widget's label contains the entered 'x' shortcut.
• const char ∗ **tooltip** () const
  Gets the current tooltip text.
• void **tooltip** (const char ∗text)
  Sets the current tooltip text.
• Fl_Window ∗ **top_window** () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window ∗ **top_window_offset** (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar **type** () const
  Gets the widget type.
• void **type** (uchar t)
  Sets the widget type.
• int **use_accents_menu** ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void ∗ **user_data** () const
  Gets the user data for this widget.
• void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.
• void **user_data** (void ∗v)
Sets the user data for this widget.

- unsigned int visible () const
  Returns whether a widget is visible.

- unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

- void visible_focus (int v)
  Modifies keyboard focus navigation.

- int visible_r () const
  Returns whether a widget and all its parents are visible.

- int w () const
  Gets the widget width.

- Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

- int x () const
  Gets the widget position in its window.

- int y () const
  Gets the widget position in its window.

- virtual ~Fl_Widget ()
  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum
  
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31

  flags possible values enumeration.
Protected Member Functions inherited from \texttt{Fl\_Light\_Button}

- \texttt{void draw () FL\_OVERRIDE}
  
  Draws the widget.

Protected Member Functions inherited from \texttt{Fl\_Button}

- \texttt{void simulate\_key\_action ()}

Protected Member Functions inherited from \texttt{Fl\_Widget}

- \texttt{void clear\_flag (unsigned int c)}
  
  Clears a flag in the flags mask.
- \texttt{void draw\_backdrop () const}
  
  If FL\_ALIGN\_IMAGE\_BACKDROP is set, the image or deimage will be drawn.
- \texttt{void draw\_box () const}
  
  Draws the widget box according its box style.
- \texttt{void draw\_box (Fl\_Boxtype t, Fl\_Color c) const}
  
  Draws a box of type \texttt{t}, of color \texttt{c} at the widget's position and size.
- \texttt{void draw\_box (Fl\_Boxtype t, int x, int y, int w, int h, Fl\_Color c) const}
  
  Draws a box of type \texttt{t}, of color \texttt{c} at the position \texttt{X,Y} and size \texttt{W,H}.
- \texttt{void draw\_focus () const}
  
  Draws a focus rectangle around the widget.
- \texttt{void draw\_focus (Fl\_Boxtype t, int X, int Y, int W, int H) const}
  
  Draws a focus rectangle around the widget.
- \texttt{void draw\_focus (Fl\_Boxtype t, int x, int y, int w, int h, Fl\_Color bg) const}
  
  Draws a focus box for the widget at the given position and size.
- \texttt{void draw\_label () const}
  
  Draws the widget's label at the defined label position.
- \texttt{void draw\_label (int, int, int, int) const}
  
  Draws the label in an arbitrary bounding box.
- \texttt{Fl\_Widget (int x, int y, int w, int h, const char \texttt{+label} = 0L)}
  
  Creates a widget at the given position and size.
- \texttt{unsigned int flags () const}
  
  Gets the widget flags mask.
- \texttt{void h (int v)}
  
  \textit{Internal use only.}
- \texttt{void set\_flag (unsigned int c)}
  
  Sets a flag in the flags mask.
- \texttt{void w (int v)}
  
  \textit{Internal use only.}
- \texttt{void x (int v)}
  
  \textit{Internal use only.}
- \texttt{void y (int v)}
  
  \textit{Internal use only.}

Static Protected Member Functions inherited from \texttt{Fl\_Button}

- \texttt{static void key\_release\_timeout (void *)}
Static Protected Attributes inherited from Fl_Button

- static Fl_Widget_Tracker * key_release_tracker = 0

The documentation for this class was generated from the following files:

- Fl_Radio_Light_Button.H
- Fl_Light_Button.cxx

12.110 Fl_Radio_Round_Button Class Reference

Inheritance diagram for Fl_Radio_Round_Button:

```
Fl_Widget
   Fl_Button
      Fl_Light_Button
         Fl_Round_Button
            Fl_Radio_Round_Button
```

Public Member Functions

- Fl_Radio_Round_Button (int X, int Y, int W, int H, const char *L=0)
  "Creates a new Fl_Radio_Button widget using the given position, size, and label string."

Public Member Functions inherited from Fl_Round_Button

- Fl_Round_Button (int x, int y, int w, int h, const char *l=0)
  "Creates a new Fl_Round_Button widget using the given position, size, and label string."

Public Member Functions inherited from Fl_Light_Button

- Fl_Light_Button (int x, int y, int w, int h, const char *l=0)
  "Creates a new Fl_Light_Button widget using the given position, size, and label string."

- int handle (int) FL_OVERRIDE
  "Handles the specified event."

Public Member Functions inherited from Fl_Button

- int clear ()
  "Same as value(0)."

- uchar compact ()
  "Return true if buttons are rendered as compact buttons."

- void compact (uchar v)
  "Decide if buttons should be rendered in compact mode."

- Fl_Boxtype down_box () const
  "Returns the current down box type, which is drawn when value() is non-zero."

- void down_box (Fl_Boxtype b)
  "Sets the down box type."
• **FL_Color** `down_color()` const
  (for backwards compatibility)

• void `down_color`(unsigned c)
  (for backwards compatibility)

• **Fl_Button**(int X, int Y, int W, int H, const char ∗L=0)
  
  The constructor creates the button using the given position, size, and label.

• int `set()`
  
  Same as `value(1)`.

• void `setonly()`
  
  Turns on this button and turns off all other radio buttons in the group (calling `value(1)` or `set()` does not do this).

• int `shortcut()` const
  
  Returns the current shortcut key for the button.

• void `shortcut`(const char ∗s)
  
  (for backwards compatibility)

• void `shortcut`(int s)
  
  Sets the shortcut key to s.

• char `value()` const
  
  Returns the current value of the button (0 or 1).

• int `value`(int v)
  
  Sets the current value of the button.

### Public Member Functions inherited from **Fl_Widget**

• void `_clear_fullscreen()`

• void `_set_fullscreen()`

• void `activate()`
  
  Activates the widget.

• unsigned int `active()` const
  
  Returns whether the widget is active.

• int `active_r()` const
  
  Returns whether the widget and all of its parents are active.

• **Fl_Align** `align()` const
  
  Gets the label alignment.

• void `align`(Fl_Align alignment)
  
  Sets the label alignment.

• long `argument()` const
  
  Gets the current user data (long) argument that is passed to the callback function.

• void `argument`(long v)
  
  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window ∗ `as_gl_window()`
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const ∗ `as_gl_window()` const

• virtual Fl_Group ∗ `as_group()`
  
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Group const ∗ `as_group()` const

• virtual Fl_Window ∗ `as_window()`
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window const ∗ `as_window()` const

• void `bind_deimage`(Fl_Image ∗img)
  
  Sets the image to use as part of the widget label when in the inactive state.

• void `bind_deimage`(int f)
Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- **void** **bind_image (Fl_Image ∗img)**  
  Sets the image to use as part of the widget label when in the active state.

- **void** **bind_image (int f)**  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype box () const**  
  Gets the box type of the widget.

- **void** **box (Fl_Boxtype new_box)**  
  Sets the box type for the widget.

- **Fl_Callback_p callback () const**  
  Gets the current callback function for the widget.

- **void** **callback (Fl_Callback ∗cb)**  
  Sets the current callback function for the widget.

- **void** **callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)**  
  Sets the current callback function and managed user data for the widget.

- **void** **callback (Fl_Callback ∗cb, void ∗p)**  
  Sets the current callback function and data for the widget.

- **void** **callback (Fl_Callback0 ∗cb)**  
  Sets the current callback function for the widget.

- **void** **callback (Fl_Callback1 ∗cb, long p=0)**  
  Sets the current callback function for the widget.

- **unsigned int changed () const**  
  Checks if the widget value changed since the last callback.

- **void** **clear_active ()**  
  Marks the widget as inactive without sending events or changing focus.

- **void** **clear_changed ()**  
  Marks the value of the widget as unchanged.

- **void** **clear_damage (uchar c=0)**  
  Clears or sets the damage flags.

- **void** **clear_output ()**  
  Sets a widget to accept input.

- **void** **clear_visible ()**  
  Hides the widget.

- **void** **clear_visible_focus ()**  
  Disables keyboard focus navigation with this widget.

- **Fl_Color color () const**  
  Gets the background color of the widget.

- **void** **color (Fl_Color bg)**  
  Sets the background color of the widget.

- **void** **color (Fl_Color bg, Fl_Color sel)**  
  Sets the background and selection color of the widget.

- **Fl_Color color2 () const**  
  For back compatibility only.

- **void** **color2 (unsigned a)**  
  For back compatibility only.

- **int** **contains (const Fl_Widget ∗w) const**  
  Checks if w is a child of this widget.

- **void** **copy_label (const char ∗new_label)**  
  Sets the current label.

- **void** **copy_tooltip (const char ∗text)**  
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image ∗ deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image ∗ deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget ∗ widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget ∗ widget, void ∗ arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image ∗ image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image ∗ image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget ∗ wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char ∗ label () const
  Gets the current label text.
• void label (const char ∗ text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char ∗ b)
Shortcut to set the label text and type in one call.

- `Fl_Color labelcolor () const`
  
  Gets the label color.

- `void labelcolor (Fl_Color c)`
  
  Sets the label color.

- `Fl_Font labelfont () const`
  
  Gets the font to use.

- `void labelfont (Fl_Font f)`
  
  Sets the font to use.

- `Fl_Fontsize labelsizelabelsize () const`
  
  Gets the font size in pixels.

- `void labelsizelabelsize (Fl_Fontsize pix)`
  
  Sets the font size in pixels.

- `Fl_Labeltype labeltype () const`
  
  Gets the label type.

- `void labeltype (Fl_Labeltype a)`
  
  Sets the label type.

- `void measure_label (int &ww, int &hh) const`
  
  Sets width ww and height hh accordingly with the label size.

- `bool needs_keyboard () const`
  
  Returns whether this widget needs a keyboard.

- `void needs_keyboard (bool needs)`
  
  Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  
  Returns if a widget is used for output only.

- `Fl_Group *parent () const`
  
  Returns a pointer to the parent widget.

- `void parent (Fl_Group *p)`
  
  Internal use only - "for hacks only".

- `void position (int X, int Y)`
  
  Repositions the window or widget.

- `void redraw ()`
  
  Schedules the drawing of the widget.

- `void redraw_label ()`
  
  Schedules the drawing of the label.

- `virtual void resize (int x, int y, int w, int h)`
  
  Changes the size or position of the widget.

- `Fl_Color selection_color () const`
  
  Gets the selection color.

- `void selection_color (Fl_Color a)`
  
  Sets the selection color.

- `void set_active ()`
  
  Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  
  Marks the value of the widget as changed.

- `void set_output ()`
  
  Sets a widget to output only.

- `void set_visible ()`
  
  Makes the widget visible.

- `void set_visible_focus ()`
  
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses ' &' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses ' &' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered ' &x' shortcut.
• const char ∗ tooltip () const
  Gets the current tooltip text.
• void tooltip (const char ∗text)
  Sets the current tooltip text.
• Fl_Window ∗ top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void ∗v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
Gets the widget position in its window.

- int y () const

  Gets the widget position in its window.

- virtual ~Fl_Widget ()

  Destroys the widget.

### Additional Inherited Members

#### Static Public Member Functions Inherited from `Fl_Widget`

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

#### Protected Types Inherited from `Fl_Widget`

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP.Relative = 1<<16, COPIED.TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

#### Protected Member Functions Inherited from `Fl_Light_Button`

- void draw () FL_OVERRIDE

  Draws the widget.

#### Protected Member Functions Inherited from `Fl_Button`

- void simulate_key_action ()

#### Protected Member Functions Inherited from `Fl_Widget`

- void clear_flag (unsigned int c)

  Clears a flag in the flags mask.

- void draw_backdrop () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void draw_box () const

  Draws the widget box according its box style.

- void draw_box (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.

- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
• void **draw_focus** () const
  Draws a focus rectangle around the widget.
• void **draw_focus** (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
• void **draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
• void **draw_label** () const
  Draws the widget's label at the defined label position.
• void **draw_label** (int, int, int) const
  Draws the label in an arbitrary bounding box.
• Fl_Widget (int x, int y, int w, int h, const char * = "0")
  Creates a widget at the given position and size.
• unsigned int **flags** () const
  Gets the widget flags mask.
• void **h** (int v)
  Internal use only.
• void **set_flag** (unsigned int c)
  Sets a flag in the flags mask.
• void **w** (int v)
  Internal use only.
• void **x** (int v)
  Internal use only.
• void **y** (int v)
  Internal use only.

Static Protected Member Functions inherited from Fl_Button

• static void **key_release_timeout** (void *)

Static Protected Attributes inherited from Fl_Button

• static Fl_Widget_Tracker * **key_release_tracker** = 0

12.110.1 Constructor & Destructor Documentation

12.110.1.1 Fl_Radio_Round_Button()

Fl_Radio_Round_Button::Fl_Radio_Round_Button (int X, int Y, int W, int H, const char * = "0")
Creates a new Fl_Radio_Button widget using the given position, size, and label string.
The button type() is set to FL_RADIO_BUTTON.

Parameters

| in | X, Y, W, H | position and size of the widget |
| in | L | widget label, default is no label |

The documentation for this class was generated from the following files:

• Fl_Radio_Round_Button.H
• Fl_Round_Button.cxx
12.111 Fl_Rect Class Reference

Rectangle with standard FLTK coordinates (X, Y, W, H).

```cpp
#include <Fl_Rect.H>
```

Public Member Functions

- **int b () const**
  gets the bottom edge (y + h).
- **void b (int B)**
  sets the height based on B and y
- **Fl_Rect ()**
  The default constructor creates an empty rectangle (x = y = w = h = 0).
- **Fl_Rect (const Fl_Widget &widget)**
  This constructor creates a rectangle based on a widget's position and size.
- **Fl_Rect (const Fl_Widget &const widget)**
  This constructor creates a rectangle based on a widget's position and size.
- **Fl_Rect (int W, int H)**
  This constructor creates a rectangle with x = y = 0 and the given width and height.
- **Fl_Rect (int X, int Y, int W, int H)**
  This constructor creates a rectangle with the given x,y coordinates and the given width and height.
- **Fl_Rect (int X, int Y, int W, int H, Fl_Boxtype bt)**
  This constructor creates a rectangle with the given x,y coordinates and the given width and height reduced by the box frame size.
- **int h () const**
  gets the height
- **void h (int H)**
  sets the height
- **void inset (Fl_Boxtype bt)**
  Move all edges in by the frame size of box type bt.
- **void inset (int d)**
  Move all edges in by d.
- **void inset (int left, int top, int right, int bottom)**
  Move all edges in by left, top, right, bottom.
- **int r () const**
  gets the right edge (x + w).
- **void r (int R)**
  sets the width based on R and x
- **int w () const**
  gets the width
- **void w (int W)**
  sets the width
- **int x () const**
  gets the x coordinate (left edge)
- **void x (int X)**
  sets the x coordinate (left edge)
- **int y () const**
  gets the y coordinate (top edge)
- **void y (int Y)**
  sets the y coordinate (top edge)
Friends

- bool operator!=(const Fl_Rect &lhs, const Fl_Rect &rhs)
- bool operator==(const Fl_Rect &lhs, const Fl_Rect &rhs)

12.111.1 Detailed Description

Rectangle with standard FLTK coordinates (X, Y, W, H).
This may be used internally, for overloaded widget constructors and other overloaded methods like \texttt{fl\_measure()}, \texttt{fl\_text\_extents()}, \texttt{fl\_rect()}, \texttt{fl\_rectf()}, and maybe more.

12.111.2 Constructor & Destructor Documentation

12.111.2.1 Fl_Rect()

\texttt{Fl\_Rect::Fl\_Rect (}
\begin{verbatim}
int X,
int Y,
int W,
int H,
Fl\_Boxtype bt ) [inline]
\end{verbatim}

This constructor creates a rectangle with the given x,y coordinates and the given width and height reduced by the box frame size.
This is the same as using the constructor w/o \texttt{bt} and subsequently calling inset(\texttt{bt}).

12.111.3 Member Function Documentation

12.111.3.1 b()

\texttt{int Fl\_Rect::b ( ) const [inline]}

gets the bottom edge (y + h).

Note

\texttt{r()} and \texttt{b()} are coordinates outside the area of the rectangle.

12.111.3.2 inset() [1/3]

\texttt{void Fl\_Rect::inset (}
\begin{verbatim}
Fl\_Boxtype bt ) [inline]
\end{verbatim}

Move all edges in by the frame size of box type \texttt{bt}.
Shrinks the rectangle at all sides by the frame width or height of the given box type \texttt{bt}.
This method uses the frame sizes given by the box type \texttt{bt} using

- Fl::box\_dx(bt)
- Fl::box\_dy(bt)
- Fl::box\_dw(bt)
- Fl::box\_dh(bt)

If the rectangle is smaller than the frame sizes the result is undefined, i.e. an invalid or empty rectangle.

12.111.3.3 inset() [2/3]

\texttt{void Fl\_Rect::inset (}
\begin{verbatim}
int d ) [inline]
\end{verbatim}

Move all edges in by \texttt{d}.
Shrinks the rectangle by \texttt{d} at all sides keeping the center of the rectangle at the same spot.
If \texttt{d} is negative, the rectangle is enlarged.
If \texttt{d} >= \texttt{w()} or \texttt{h()} the result is undefined, i.e. an invalid or empty rectangle.
12.111.3.4 inset() [3/3]

```cpp
void Fl_Rect::inset (  
    int left,  
    int top,  
    int right,  
    int bottom ) [inline]
```

Move all edges in by `left`, `top`, `right`, `bottom`.
Shrinks the rectangle on all sides keeping the center of the rectangle at the same spot.
If any value is negative, the rectangle is enlarged.
Values are not range checked; it is possible to create an invalid or empty rectangle.

12.111.3.5 r()

```cpp
int Fl_Rect::r ( ) const [inline]
```
gets the right edge `(x + w)`.

Note

`r()` and `b()` are coordinates **outside** the area of the rectangle.

The documentation for this class was generated from the following file:

- Fl_Rect.H

### 12.112 Fl_Scroll::Fl_Region_LRTB Struct Reference

A local struct to manage a region defined by left/right/top/bottom.

```cpp
#include <Fl_Scroll.H>
```

**Public Attributes**

- int b
  - (b)ottom "y" position, aka `y2`
- int l
  - (l)eft "x" position, aka `x1`
- int r
  - (r)ight "x" position, aka `x2`
- int t
  - (t)op "y" position, aka `y1`

### 12.112.1 Detailed Description

A local struct to manage a region defined by left/right/top/bottom.
The documentation for this struct was generated from the following file:

- Fl_Scroll.H

### 12.113 Fl_Scroll::Fl_Region_XYWH Struct Reference

A local struct to manage a region defined by xywh.

```cpp
#include <Fl_Scroll.H>
```

**Public Attributes**

- int h
- int w
- int x
- int y
12.114 Fl_Repeat_Button Class Reference

The Fl_Repeat_Button is a subclass of Fl_Button that generates a callback when it is pressed and then repeatedly generates callbacks as long as it is held down.

#include <Fl_Repeat_Button.H>

Inheritance diagram for Fl_Repeat_Button:

```
Fl_Widget
   v
Fl_Button
   v
Fl_Repeat_Button
```

Public Member Functions

- void deactivate ()
- Fl_Repeat_Button (int X, int Y, int W, int H, const char *l=0)
  Creates a new Fl_Repeat_Button widget using the given position, size, and label string.
- int handle (int) FL_OVERRIDE
  Handles the specified event.

Public Member Functions inherited from Fl_Button

- int clear ()
  Same as value(0).
- uchar compact ()
  Return true if buttons are rendered as compact buttons.
- void compact (uchar v)
  Decide if buttons should be rendered in compact mode.
- Fl_Boxtype down_box () const
  Returns the current down box type, which is drawn when value() is non-zero.
- void down_box (Fl_Boxtype b)
  Sets the down box type.
- Fl_Color down_color () const
  (for backwards compatibility)
- void down_color (unsigned c)
  (for backwards compatibility)
- Fl_Button (int X, int Y, int W, int H, const char *L=0)
  The constructor creates the button using the given position, size, and label.
- int set ()
  Same as value(1).
- void setonly ()
  Turns on this button and turns off all other radio buttons in the group (calling value(1) or set() does not do this).
- int shortcut () const
Returns the current shortcut key for the button.

- void **shortcut** (const char ∗s)  
  (for backwards compatibility)
- void **shortcut** (int s)  
  Sets the shortcut key to s.
- char **value** () const  
  Returns the current value of the button (0 or 1).
- int **value** (int v)  
  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void **activate** ()  
  Activates the widget.
- unsigned int **active** () const  
  Returns whether the widget is active.
- int **active_r** () const  
  Returns whether the widget and all of its parents are active.
- **Fl_Align align** () const  
  Gets the label alignment.
- void **align** (Fl_Align alignment)  
  Sets the label alignment.
- long **argument** () const  
  Gets the current user data (long) argument that is passed to the callback function.
- void **argument** (long v)  
  Sets the current user data (long) argument that is passed to the callback function.
- virtual **class Fl_Gl_Window** ∗**as_gl_window** ()  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual **class Fl_Gl_Window** const ∗**as_gl_window** () const  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual **class Fl_Group** ∗**as_group** ()  
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual **class Fl_Group** const ∗**as_group** () const  
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual **class Fl_Window** ∗**as_window** ()  
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual **class Fl_Window** const ∗**as_window** () const  
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- void **bind_deimage** (Fl_Image ∗img)  
  Sets the image to use as part of the widget label when in the inactive state.
- void **bind_deimage** (int f)  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void **bind_image** (Fl_Image ∗img)  
  Sets the image to use as part of the widget label when in the active state.
- void **bind_image** (int f)  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- **Fl_Boxtype box** () const  
  Gets the box type of the widget.
- void **box** (Fl_Boxtype new_box)  
  Sets the box type for the widget.
- **Fl_Callback_p** callback () const  
  Gets the current callback function for the widget.
• void callback (Fl_Callback *cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

• void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.

• void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.

• unsigned int changed () const
  Checks if the widget value changed since the last callback.

• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

• void clear_changed ()
  Marks the value of the widget as unchanged.

• void clear_damage (uchar c=0)
  Clears or sets the damage flags.

• void clear_output ()
  Sets a widget to accept input.

• void clear_visible ()
  Hides the widget.

• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

• Fl_Color color () const
  Gets the background color of the widget.

• void color (Fl_Color bg)
  Sets the background color of the widget.

• void color (Fl_Color bg, Fl_Color sel)
  Gets the background and selection color of the widget.

• Fl_Color color2 () const
  For back compatibility only.

• void color2 (unsigned a)
  For back compatibility only.

• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.

• void copy_label (const char *new_label)
  Sets the current label.

• void copy_tooltip (const char *text)
  Sets the current tooltip text.

• uchar damage () const
  Returns non-zero if draw() needs to be called.

• void damage (uchar c)
  Sets the damage bits for the widget.

• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

• int damage_resize (int, int, int, int)
  Internal use only.

• void deactivate ()
  Deactivates the widget.

• Fl_Image *deimage ()
Gets the image that is used as part of the widget label when in the inactive state.

- `const Fl_Image * deimage () const`
  
  Gets the image that is used as part of the widget label when in the inactive state.

- `void deimage (Fl_Image &img)`
  
  Sets the image to use as part of the widget label when in the inactive state.

- `void deimage (Fl_Image *img)`
  
  Sets the image to use as part of the widget label when in the inactive state.

- `int deimage_bound () const`
  
  Returns whether the inactive image is managed by the widget.

- `void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  
  Calls the widget callback function with default arguments.

- `void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  
  Calls the widget callback function with arbitrary arguments.

- `void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  
  Calls the widget callback function with arbitrary arguments.

- `void draw_label (int, int, int, int, Fl_Align) const`
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- `int h () const`
  
  Gets the widget height.

- `virtual void hide ()`
  
  Makes a widget invisible.

- `Fl_Image * image ()`
  
  Gets the image that is used as part of the widget label when in the active state.

- `const Fl_Image * image () const`
  
  Gets the image that is used as part of the widget label when in the active state.

- `void image (Fl_Image &img)`
  
  Sets the image to use as part of the widget label when in the active state.

- `void image (Fl_Image *img)`
  
  Sets the image to use as part of the widget label when in the active state.

- `int image_bound () const`
  
  Returns whether the image is managed by the widget.

- `int inside (const Fl_Widget *wgt) const`
  
  Checks if this widget is a child of wgt.

- `int is_label_copied () const`
  
  Returns whether the current label was assigned with copy_label().

- `const char * label () const`
  
  Gets the current label text.

- `void label (const char *text)`
  
  Sets the current label pointer.

- `void label (Fl_Labeltype a, const char *b)`
  
  Shortcut to set the label text and type in one call.

- `Fl_Color labelcolor () const`
  
  Gets the label color.

- `void labelcolor (Fl_Color c)`
  
  Sets the label color.

- `Fl_Font labelfont () const`
  
  Gets the font to use.

- `void labelfont (Fl_Font f)`
  
  Sets the font to use.

- `Fl_Fontsize labelsize () const`
  
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.

• Fl_Labeltype labeltype () const
  Gets the label type.

• void labeltype (Fl_Labeltype a)
  Sets the label type.

• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.

• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.

• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.

• unsigned int output () const
  Returns if a widget is used for output only.

• Fl_Group * parent () const
  Returns a pointer to the parent widget.

• void parent (Fl_Group *p)
  Internal use only - "for hacks only".

• void position (int X, int Y)
  Repositions the window or widget.

• void redraw ()
  Schedules the drawing of the widget.

• void redraw_label ()
  Schedules the drawing of the label.

• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.

• Fl_Color selection_color () const
  Gets the selection color.

• void selection_color (Fl_Color a)
  Sets the selection color.

• void set_active ()
  Marks the widget as active without sending events or changing focus.

• void set_changed ()
  Marks the value of the widget as changed.

• void set_output ()
  Sets a widget to output only.

• void set_visible ()
  Makes the widget visible.

• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.

• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.

• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

• virtual void show ()
  Makes a widget visible.

• void size (int W, int H)
  Changes the size of the widget.

• int take_focus ()
  Gives the widget the keyboard focus.

• unsigned int takesevents () const
Returns if the widget is able to take events.

- int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.

- const char * tooltip () const
  Gets the current tooltip text.

- void tooltip (const char *text)
  Sets the current tooltip text.

- Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.

- Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const
  Gets the widget type.

- void type (uchar t)
  Sets the widget type.

- int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void * user_data () const
  Gets the user data for this widget.

- void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

- void user_data (void *v)
  Sets the user data for this widget.

- unsigned int visible () const
  Returns whether a widget is visible.

- unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

- void visible_focus (int v)
  Modifies keyboard focus navigation.

- int visible_r () const
  Returns whether a widget and all its parents are visible.

- int w () const
  Gets the widget width.

- Fl_When when () const
  Returns the conditions under which the callback is called.

- void when (uchar i)
  Sets the flags used to decide when a callback is called.

- Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

- int x () const
  Gets the widget position in its window.

- int y () const
  Gets the widget position in its window.

- virtual ~Fl_Widget ()
  Destroys the widget.
Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback(Fl_Widget *widget, void *data)
  
  The default callback for all widgets that don't set a callback.

- static unsigned int labelShortcut(const char *t)
  
  Returns the Unicode value of the '&x' shortcut in a given text.

- static int testShortcut(const char *t, const bool require_alt=false)
  
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

flags possible values enumeration.

Protected Member Functions inherited from Fl_Button

- void draw() FL_OVERRIDE
  
  Draws the widget.

- void simulate_key_action()
Draws the label in an arbitrary bounding box.

- **Fl_Widget** (int x, int y, int w, int h, const char ∗label=0L)
  
  Creates a widget at the given position and size.

- unsigned int flags () const
  
  Gets the widget flags mask.

- void h (int v)
  
  Internal use only.

- void set_flag (unsigned int c)
  
  Sets a flag in the flags mask.

- void w (int v)
  
  Internal use only.

- void x (int v)
  
  Internal use only.

- void y (int v)
  
  Internal use only.

**Static Protected Member Functions inherited from Fl_Button**

- static void key_release_timeout (void ∗)

**Static Protected Attributes inherited from Fl_Button**

- static Fl_Widget_Tracker ∗key_release_tracker = 0

### Detailed Description

The Fl_REPEAT_Button is a subclass of Fl_Button that generates a callback when it is pressed and then repeatedly generates callbacks as long as it is held down. The speed of the repeat is fixed and depends on the implementation.

### Constructor & Destructor Documentation

#### Fl_REPEAT_Button()

Fl_REPEAT_Button::Fl_REPEAT_Button {  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char ∗l = 0  
}  

Creates a new Fl_REPEAT_Button widget using the given position, size, and label string. The default boxtype is FL_UP_BOX. Deletes the button.

### Member Function Documentation

#### handle()

int Fl_REPEAT_Button::handle {  
  int event  
} [virtual]

Handles the specified event. You normally don’t call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited handle() method in your overridden method so that you don’t short-circuit events that you don’t handle. In this last case you should return the callee retval. One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in event | the kind of event received |

Return values

| 0 | if the event was not used or understood |
| 1 | if the event was used and can be deleted |

See also

FL_Event

Reimplemented from Fl_Button.

The documentation for this class was generated from the following files:

- Fl_Repeat_Button.H
- Fl_Repeat_Button.hxx

### 12.115 Fl_Return_Button Class Reference

The Fl_Return_Button is a subclass of Fl_Button that generates a callback when it is pressed or when the user presses the Enter key.

```
#include <Fl_Return_Button.H>
```

Inheritance diagram for Fl_Return_Button:

```
Fl_Widget
  Fl_Button
  Fl_Return_Button
```

**Public Member Functions**

- `Fl_Return_Button(int X, int Y, int W, int H, const char *l=0)`
  
  Creates a new Fl_Return_Button widget using the given position, size, and label string.

- `int handle(int)` FL_OVERRIDE
  
  Handles the specified event.

**Public Member Functions inherited from Fl_Button**

- `int clear()` 
  
  Same as value(0).

- `uchar compact()` 
  
  Return true if buttons are rendered as compact buttons.

- `void compact(uchar v)`
Decide if buttons should be rendered in compact mode.

- **Fl_Boxtype down_box () const**
  Returns the current down box type, which is drawn when value() is non-zero.

- **void down_box (Fl_Boxtype b)**
  Sets the down box type.

- **Fl_Color down_color () const**
  (for backwards compatibility)

- **void down_color (unsigned c)**
  (for backwards compatibility)

- **Fl_Button (int X, int Y , int W, int H, const char ∗L=0)**
  The constructor creates the button using the given position, size, and label.

- **int set ()**
  Same as value(1).

- **void setonly ()**
  Turns on this button and turns off all other radio buttons in the group (calling value(1) or set() does not do this).

- **int shortcut () const**
  Returns the current shortcut key for the button.

- **void shortcut (const char ∗s)**
  (for backwards compatibility)

- **void shortcut (int s)**
  Sets the shortcut key to s.

- **char value () const**
  Returns the current value of the button (0 or 1).

- **int value (int v)**
  Sets the current value of the button.

**Public Member Functions inherited from Fl_Widget**

- **void _clear_fullscreen ()**

- **void _set_fullscreen ()**

- **void activate ()**
  Activates the widget.

- **unsigned int active () const**
  Returns whether the widget is active.

- **int active_r () const**
  Returns whether the widget and all of its parents are active.

- **Fl_Align align () const**
  Gets the label alignment.

- **void align (Fl_Align alignment)**
  Sets the label alignment.

- **long argument () const**
  Gets the current user data (long) argument that is passed to the callback function.

- **void argument (long v)**
  Sets the current user data (long) argument that is passed to the callback function.

- **virtual class Fl_Gl_Window ∗ as_gl_window ()**
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- **virtual class Fl_Gl_Window const ∗ as_gl_window () const**

- **virtual Fl_Group ∗ as_group ()**
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- **virtual Fl_Group const ∗ as_group () const**

- **virtual Fl_Window ∗ as_window ()**

Generated by Doxygen
Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const * as_window () const
  
  Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage (Fl_Image *img)
  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_deimage (int f)
  
  Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)
  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- Fl_Boxtype box () const
  
  Gets the box type of the widget.

- void box (Fl_Boxtype new_box)
  
  Sets the box type for the widget.

- Fl_Callback * callback () const
  
  Gets the current callback function for the widget.

- void callback (Fl_Callback cb)
  
  Sets the current callback function for the widget.

- void callback (Fl_Callback cb, Fl_Callback_User_Data *p, bool auto_free)
  
  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback cb, void *p)
  
  Sets the current callback function and data for the widget.

- void callback (Fl_Callback0 cb)
  
  Sets the current callback function for the widget.

- void callback (Fl_Callback1 cb, long p=0)
  
  Sets the current callback function for the widget.

- unsigned int changed () const
  
  Checks if the widget value changed since the last callback.

- void clear_active ()
  
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  
  Clears or sets the damage flags.

- void clear_output ()
  
  Sets a widget to accept input.

- void clear_visible ()
  
  Hides the widget.

- void clear_visible_focus ()
  
  Disables keyboard focus navigation with this widget.

- Fl_Color color () const
  
  Gets the background color of the widget.

- void color (Fl_Color bg)
  
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  
  Sets the background and selection color of the widget.

- Fl_Color color2 () const
  
  For back compatibility only.

- void color2 (unsigned a)
  
  For back compatibility only.

- int contains (const Fl_Widget *w) const
Checks if \( w \) is a child of this widget.

- **void copy_label** (const char *new_label)
  Sets the current label.

- **void copy_tooltip** (const char *text)
  Sets the current tooltip text.

- **uchar damage** () const
  Returns non-zero if draw() needs to be called.

- **void damage** (uchar c)
  Sets the damage bits for the widget.

- **void damage** (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- **int damage_resize**(int, int, int, int)
  Internal use only.

- **void deactivate** ()
  Deactivates the widget.

- **Fl_Image * deimage** ()
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image * deimage** () const
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage** (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage** (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound** () const
  Returns whether the inactive image is managed by the widget.

- **void do_callback** (Fl_Callback_REASON reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

- **void do_callback** (Fl_Widget *widget, long arg, Fl_Callback_REASON reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- **void do_callback** (Fl_Widget *widget, void *arg=0, Fl_Callback_REASON reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- **void draw_label** (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h** () const
  Gets the widget height.

- virtual **void hide** ()
  Makes a widget invisible.

- **Fl_Image * image** ()
  Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image * image** () const
  Gets the image that is used as part of the widget label when in the active state.

- **void image** (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.

- **void image** (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound** () const
  Returns whether the image is managed by the widget.

- **int inside** (const Fl_Widget *wgt) const
  Checks if this widget is a child of \( wgt \).

- **int is_label_copied** () const
  Returns whether the current label was assigned with copy_label().
• `const char * label () const`
  
  Gets the current label text.

• `void label (const char *text)`
  
  Sets the current label pointer.

• `void label (Fl_Labeltype a, const char *b)`

  Shortcut to set the label text and type in one call.

• `Fl_Color labelcolor () const`

  Gets the label color.

• `void labelcolor (Fl_Color c)`

  Sets the label color.

• `Fl_Font labelfont () const`

  Gets the font to use.

• `void labelfont (Fl_Font f)`

  Sets the font to use.

• `Fl_Fontsize labelsize () const`

  Gets the font size in pixels.

• `void labelsize (Fl_Fontsize pix)`

  Sets the font size in pixels.

• `Fl_Labeltype labelttype () const`

  Gets the label type.

• `void labelttype (Fl_Labeltype a)`

  Sets the label type.

• `void measure_label (int &ww, int &hh) const`

  Sets width `ww` and height `hh` accordingly with the label size.

• `bool needs_keyboard () const`

  Returns whether this widget needs a keyboard.

• `void needs_keyboard (bool needs)`

  Sets whether this widget needs a keyboard.

• `unsigned int output () const`

  Returns if a widget is used for output only.

• `Fl_Group * parent () const`

  Returns a pointer to the parent widget.

• `void parent (Fl_Group *p)`

  Internal use only - “for hacks only”.

• `void position (int X, int Y)`

  Repositions the window or widget.

• `void redraw ()`

  Schedules the drawing of the widget.

• `void redraw_label ()`

  Schedules the drawing of the label.

• `virtual void resize (int x, int y, int w, int h)`

  Changes the size or position of the widget.

• `Fl_Color selection_color () const`

  Gets the selection color.

• `void selection_color (Fl_Color a)`

  Sets the selection color.

• `void set_active ()`

  Marks the widget as active without sending events or changing focus.

• `void set_changed ()`

  Marks the value of the widget as changed.

• `void set_output ()`
Sets a widget to output only.

- void set_visible ()
  Makes the widget visible.

- void set_visible_focus ()
  Enables keyboard focus navigation with this widget.

- int shortcut_label () const
  Returns whether the widget's label uses ' & ' to indicate shortcuts.

- void shortcut_label (int value)
  Sets whether the widget's label uses ' & ' to indicate shortcuts.

- virtual void show ()
  Makes a widget visible.

- void size (int W, int H)
  Changes the size of the widget.

- int take_focus ()
  Gives the widget the keyboard focus.

- unsigned int takesevents () const
  Returns if the widget is able to take events.

- int testShortcut ()
  Returns true if the widget's label contains the entered ' & X ' shortcut.

- const char *tooltip () const
  Gets the current tooltip text.

- void tooltip (const char *text)
  Sets the current tooltip text.

- Fl_Window *top_window () const
  Returns a pointer to the top-level window for the widget.

- Fl_Window *top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const
  Gets the widget type.

- void type (uchar t)
  Sets the widget type.

- int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void *user_data () const
  Gets the user data for this widget.

- void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

- void user_data (void *v)
  Sets the user data for this widget.

- unsigned int visible () const
  Returns whether a widget is visible.

- unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

- void visible_focus (int v)
  Modifies keyboard focus navigation.

- int visible_r () const
  Returns whether a widget and all its parents are visible.

- int w () const
  Gets the widget width.

- Fl_When when () const
  Returns the conditions under which the callback is called.
• void \texttt{when} (uchar \texttt{i})
  \textit{Sets the flags used to decide when a callback is called.}

• \texttt{Fl\_Window * window () const}
  \textit{Returns a pointer to the nearest parent window up the widget hierarchy.}

• \texttt{int x () const}
  \textit{Gets the widget position in its window.}

• \texttt{int y () const}
  \textit{Gets the widget position in its window.}

• virtual \texttt{~Fl\_Widget ()}
  \textit{Destroys the widget.}

\textbf{Protected Member Functions}

• void \texttt{draw () FL\_OVERRIDE}
  \textit{Draws the widget.}

\textbf{Protected Member Functions inherited from Fl\_Button}

• void \texttt{simulate\_key\_action ()}

\textbf{Protected Member Functions inherited from Fl\_Widget}

• void \texttt{clear\_flag} (unsigned int \texttt{c})
  \textit{Clears a flag in the flags mask.}

• void \texttt{draw\_backdrop () const}
  \textit{If FL\_ALIGN\_IMAGE\_BACKDROP is set, the image or deimage will be drawn.}

• void \texttt{draw\_box () const}
  \textit{Draws the widget box according its box style.}

• void \texttt{draw\_box (Fl\_Boxtype t, Fl\_Color c) const}
  \textit{Draws a box of type t, of color c at the widget's position and size.}

• void \texttt{draw\_box (Fl\_Boxtype t, int x, int y, int w, int h, Fl\_Color c) const}
  \textit{Draws a box of type t, of color c at the position X,Y and size W,H.}

• void \texttt{draw\_focus () const}
  \textit{Draws a focus rectangle around the widget.}

• void \texttt{draw\_focus (Fl\_Boxtype t, int X, int Y, int W, int H) const}
  \textit{Draws a focus rectangle around the widget.}

• void \texttt{draw\_focus (Fl\_Boxtype t, int x, int y, int w, int h, Fl\_Color bg) const}
  \textit{Draws a focus box for the widget at the given position and size.}

• void \texttt{draw\_label () const}
  \textit{Draws the widget's label at the defined label position.}

• void \texttt{draw\_label (int, int, int, int) const}
  \textit{Draws the label in an arbitrary bounding box.}

• \texttt{Fl\_Widget (int x, int y, int w, int h, const char *label=0L)}
  \textit{Creates a widget at the given position and size.}

• unsigned int \texttt{flags () const}
  \textit{Gets the widget flags mask.}

• void \texttt{h (int v)}
  \textit{Internal use only.}

• void \texttt{set\_flag} (unsigned int \texttt{c})
  \textit{Sets a flag in the flags mask.}

• void \texttt{w (int v)}
  \textit{Internal use only.}
• void x (int v)
  *Internal use only.*
• void y (int v)
  *Internal use only.*

### Additional Inherited Members

#### Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget ∗widget, void ∗data)
  *The default callback for all widgets that don't set a callback.*
• static unsigned int label_shortcut (const char ∗t)
  *Returns the Unicode value of the 'x' shortcut in a given text.*
• static int test_shortcut (const char ∗t, const bool require_alt=false)
  *Returns true if the given text t contains the entered 'x' shortcut.*

### Protected Types inherited from Fl_Widget

```cpp
enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
    = 1<<19, NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
};
```

*flags possible values enumeration.*

#### Static Protected Member Functions inherited from Fl_Button

• static void key_release_timeout (void ∗)

#### Static Protected Attributes inherited from Fl_Button

• static Fl_Widget_Tracker ∗key_release_tracker = 0

### 12.115.1 Detailed Description

The Fl_Return_Button is a subclass of Fl_Button that generates a callback when it is pressed or when the user presses the Enter key.

A carriage-return symbol is drawn next to the button label.

![Fl_Return_Button](image)

*Figure 12.37 Fl_Return_Button*

### 12.115.2 Constructor & Destructor Documentation

#### 12.115.2.1 Fl_Return_Button()

```cpp
Fl_Return_Button::Fl_Return_Button (int X, ...
```
int Y,
int W,
int H,
const char * l = 0)

Creates a new Fl_Return_Button widget using the given position, size, and label string.
The default boxtype is FL_UP_BOX.
The inherited destructor deletes the button.

12.115.3 Member Function Documentation

12.115.3.1 draw()

void Fl_Return_Button::draw ( ) [protected], [virtual]

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as
soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded
scrollbar, you can do it (because draw() is virtual) like this:
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
Reimplemented from Fl_Button.

12.115.3.2 handle()

int Fl_Return_Button::handle ( 
    int event ) [virtual]

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-
circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class.
This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The
effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses)
in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In
such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by
returning 1 even if you didn't do anything with it.

Parameters

|  in  | event | the kind of event received |

Return values

| 0  | if the event was not used or understood |
| 1  | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Button.
The documentation for this class was generated from the following files:

- Fl_Return_Button.H
- Fl_Return_Button.cxx

Generated by Doxygen
12.116 FL_RGB_Image Class Reference

The **FL_RGB_Image** class supports caching and drawing of full-color images with 1 to 4 channels of color information.

```c
#include <FL/Image.H>
```

Inheritance diagram for FL_RGB_Image:

```
Fl_RGB_Image
  └ Fl_Image
    └ Fl_BMP_Image
        └ Fl_JPEG_Image
        └ Fl_PNG_Image
        └ Fl_PNM_Image
        └ Fl_SVG_Image
```

### Public Member Functions

- `virtual Fl_SVG_Image* as_svg_image()`  
  Returns whether an image is an **Fl_SVG_Image** or not.
- `int cache_h()`  
- `int cache_w()`  
- `void color_average(Fl_Color c, float i) FL_OVERRIDE`
  The `color_average()` method averages the colors in the image with the provided FLTK color value.
- `Fl_Image* copy()` const  
- `Fl_Image* copy(int W, int H) const FL_OVERRIDE`
  Creates a resized copy of the image.
- `void desaturate() FL_OVERRIDE`
  The `desaturate()` method converts an image to grayscale.
- `void draw(int X, int Y)`  
- `void draw(int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE`
  Draws the image to the current drawing surface with a bounding box.
- `Fl_RGB_Image(const Fl_Pixmap* pxm, Fl_Color bg=FL_GRAY)`  
  The constructor creates a new RGBA image from the specified **Fl_Pixmap**.
- `Fl_RGB_Image(const uchar* bits, int bits_length, int W, int H, int D, int LD)`  
  The constructor creates a new image from the specified data.
- `Fl_RGB_Image(const uchar* bits, int W, int H, int D=3, int LD=0)`  
  The constructor creates a new image from the specified data.
- `void label(Fl_Menu_Item* m) FL_OVERRIDE`
  This method is an obsolete way to set the image attribute of a menu item.
- `void label(Fl_Widget* w) FL_OVERRIDE`
  This method is an obsolete way to set the image attribute of a widget or menu item.
- `virtual void normalize()`  
  Makes sure the object is fully initialized.
- `void uncache() FL_OVERRIDE`
  If the image has been cached for display, delete the cache data.
- `~FL_RGB_Image() FL_OVERRIDE`
  The destructor frees all memory and server resources that are used by the image.
Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image ∗ as_shared_image ()
  Returns whether an image is an Fl_Shared_Image or not.
- Fl_image ∗ copy () const
  Creates a copy of the image in the same size.
- int count () const
  Returns the number of data values associated with the image.
- int d () const
  Returns the image depth.
- const char ∗ ∗ data () const
  Returns a pointer to the current image data array.
- int data_h () const
  Returns the height of the image data.
- int data_w () const
  Returns the width of the image data.
- void draw (int X, int Y)
  Draws the image to the current drawing surface.
- int fail () const
  Returns a value that is not 0 if there is currently no image available.
- Fl_image (int W, int H, int D)
  The constructor creates an empty image with the specified width, height, and depth.
- int h () const
  Returns the current image drawing height in FLTK units.
- void inactive ()
  The inactive() method calls color_average(FL_BACKGROUNDCOLOR, 0.33f) to produce an image that appears grayed out.
- int ld () const
  Returns the current line data size in bytes.
- virtual void release ()
  Releases an Fl_image - the same as 'delete this'.
- virtual void scale (int width, int height, int proportional=1, int can_expand=0)
  Sets the drawing size of the image.
- int w () const
  Returns the current image drawing width in FLTK units.
- virtual ∼Fl_image ()
  The destructor is a virtual method that frees all memory used by the image.

Static Public Member Functions

- static size_t max_size ()
  Returns the maximum allowed image size in bytes when creating an Fl_RGB_image object.
- static void max_size (size_t size)
  Sets the maximum allowed image size in bytes when creating an Fl_RGB_image object.

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype define_FL_IMAGE_LABEL ()
- static Fl_RGB_Scaling RGB_scaling ()
  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)
  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling scaling_algorithm ()
Gets what algorithm is used when resizing a source image to draw it.
• static void scaling_algorithm (FL_RGB_Scaling algorithm)

Sets what algorithm is used when resizing a source image to draw it.

Public Attributes
• int alloc_array
  If non-zero, the object's data array is delete[]d when deleting the object.
• const uchar * array
  Points to the start of the object's data array.

Friends
• class Fl_Graphics_Driver

Additional Inherited Members

Static Public Attributes inherited from Fl_Image
• static const int ERR_FILE_ACCESS = -2
• static const int ERR_FORMAT = -3
• static const int ERR_MEMORY_ACCESS = -4
• static const int ERR_NO_IMAGE = -1
• static bool register_images_done = false
  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_Image
• void d (int D)
  Sets the current image depth.
• void data (const char * const * p, int c)
  Sets the current data pointer and count of pointers in the array.
• void draw_empty (int X, int Y)
  The protected method draw_empty() draws a box with an X in it.
• int draw_scaled (int X, int Y, int W, int H)
  Draw the image to the current drawing surface rescaled to a given width and height.
• void h (int H)
  Sets the height of the image data.
• void ld (int LD)
  Sets the current line data size in bytes.
• void w (int W)
  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image
• static void labeltype (const Fl_Label * lo, int lx, int ly, int lw, int lh, Fl_Align la)
• static void measure (const Fl_Label * lo, int & lw, int & lh)

12.116.1 Detailed Description
The Fl_RGB_Image class supports caching and drawing of full-color images with 1 to 4 channels of color information.
Images with an even number of channels are assumed to contain alpha information, which is used to blend the image with the contents of the screen.
Fl_RGB_Image is defined in <FL/Fl_Image.H>, however for compatibility reasons <FL/FL_RGB_Image.H> should be included.
12.116.2 Constructor & Destructor Documentation

12.116.2.1 Fl_RGB_Image() [1/3]

The constructor creates a new image from the specified data.
The data array `bits` must contain sufficient data to provide `W * H * D` image bytes and optional line padding, see `LD`.
`W` and `H` are the width and height of the image in pixels, resp.
`D` is the image depth and can be:

- D=1: each uchar in `bits[]` is a grayscale pixel value
- D=2: each uchar pair in `bits[]` is a grayscale + alpha pixel value
- D=3: each uchar triplet in `bits[]` is an R/G/B pixel value
- D=4: each uchar quad in `bits[]` is an R/G/B/A pixel value

`LD` specifies the line data size of the array, see Fl_Image::ld(int). If `LD` is zero, then `W * D` is assumed, otherwise `LD` must be greater than or equal to `W * D` to account for (unused) extra data per line (padding).
The caller is responsible that the image data array `bits` persists as long as the image is used.
This constructor sets Fl_RGB_Image::alloc_array to 0. To have the image object control the deallocation of the data array `bits`, set alloc_array to non-zero after construction.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>bits</th>
<th>The image data array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>W</td>
<td>The width of the image in pixels.</td>
</tr>
<tr>
<td>in</td>
<td>H</td>
<td>The height of the image in pixels.</td>
</tr>
<tr>
<td>in</td>
<td>D</td>
<td>The image depth, or 'number of channels' (default=3).</td>
</tr>
<tr>
<td>in</td>
<td>LD</td>
<td>Line data size (default=0).</td>
</tr>
</tbody>
</table>

See also

Fl_Image::data(), Fl_Image::w(), Fl_Image::h(), Fl_Image::d(), Fl_Image::ld(int)

12.116.2.2 Fl_RGB_Image() [2/3]

The constructor creates a new image from the specified data.
If the provided array is too small to contain all the image data, the constructor will not generate the image to avoid illegal memory read access and instead set data to NULL and ld to ERR_MEMORY_ACCESS.

Parameters

<table>
<thead>
<tr>
<th>bits</th>
<th>image data</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits_length</td>
<td>length of the bits array in bytes</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W )</td>
<td>image width in pixels</td>
</tr>
<tr>
<td>( H )</td>
<td>image height in pixels</td>
</tr>
<tr>
<td>( D )</td>
<td>image depth in bytes, 1 for gray scale, 2 for gray with alpha, 3 for RGB, and 4 for RGB plus alpha</td>
</tr>
<tr>
<td>( LD )</td>
<td>line length in bytes, or 0 to use ( W \times D ).</td>
</tr>
</tbody>
</table>

See also

\texttt{Fl_RGB\_Image(const \texttt{uchar} * \texttt{bits}, int \( W \), int \( H \), int \( D \), int \( LD \))}

12.116.2.3 \texttt{Fl\_RGB\_Image()} [3/3]

\texttt{Fl\_RGB\_Image::Fl\_RGB\_Image (}
  \texttt{const Fl\_Pixmap *} \texttt{pxm},
  \texttt{Fl\_Color bg = FL\_GRAY})

The constructor creates a new RGBA image from the specified \texttt{Fl\_Pixmap}.
The RGBA image is built fully opaque except for the transparent area of the pixmap that is assigned the \texttt{bg} color with full transparency.
This constructor creates a new internal data array and sets \texttt{Fl\_RGB\_Image::alloc\_array} to 1 so the data array is deleted when the image is destroyed.

12.116.3 Member Function Documentation

12.116.3.1 \texttt{as\_svg\_image()}

\texttt{virtual Fl\_SVG\_Image *} \texttt{Fl\_RGB\_Image::as\_svg\_image ( ) [inline], [virtual]}

Returns whether an image is an \texttt{Fl\_SVG\_Image} or not.
This virtual method returns a pointer to the \texttt{Fl\_SVG\_Image} if this object is an instance of \texttt{Fl\_SVG\_Image} or NULL if not.
Reimplemented in \texttt{Fl\_SVG\_Image}.

12.116.3.2 \texttt{color\_average()}

\texttt{void Fl\_RGB\_Image::color\_average (}
  \texttt{Fl\_Color c,}
  \texttt{float \( i \) [virtual]}

The \texttt{color\_average()} method averages the colors in the image with the provided FLTK color value.
The first argument specifies the FLTK color to be used.
The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from \texttt{Fl\_Image}.
Reimplemented in \texttt{Fl\_SVG\_Image}.

12.116.3.3 \texttt{copy()}

\texttt{Fl\_Image *} \texttt{Fl\_RGB\_Image::copy (}
  \texttt{int \( W \),}
  \texttt{int \( H \) const [virtual]}

Creates a resized copy of the image.
The new image should be released when you are done with it.
Note: since FLTK 1.4.0 you can use \texttt{Fl\_Image::release()} for all types of images (i.e. all subclasses of \texttt{Fl\_Image}) instead of operator \texttt{delete} for \texttt{Fl\_Image\'s} and \texttt{Fl\_Image::release()} for \texttt{Fl\_Shared\_Image\'s}.
The new image data will be converted to the requested size. RGB images are resized using the algorithm set by \texttt{Fl\_Image::RGB\_scaling()}. For the new image the following equations are true:
\section*{12.116 Fl_RGB_Image Class Reference}

- \texttt{w()} == \texttt{data_w()} == W
- \texttt{h()} == \texttt{data_h()} == H

\textbf{Parameters}

\begin{tabular}{ll}
\hline
\texttt{in} & \texttt{W,H} \texttt{Requested width and height of the new image} \\
\hline
\end{tabular}

\textbf{Note}

The returned image can be safely cast to the same image type as that of the source image provided this type is one of \texttt{Fl_RGB_Image}, \texttt{Fl_SVG_Image}, \texttt{Fl_Pixmap}, \texttt{Fl_Bitmap}, \texttt{Fl_Tiled_Image}, \texttt{Fl_Anim_GIF_Image} and \texttt{Fl_Shared_Image}. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an \texttt{Fl_GIF_Image} is an object of class \texttt{Fl_Pixmap}.

Since FLTK 1.4.0 this method is \texttt{const}. If you derive your own class from \texttt{Fl_Image} or any subclass your overridden methods of \texttt{`Fl_Image::copy() const'} and \texttt{`Fl_Image::copy(int, int) const'} must also be \texttt{`const'} for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been \texttt{`const'}.

Reimplemented from \texttt{Fl_Image}.

Reimplemented in \texttt{Fl_SVG_Image}.

\subsection*{12.116.3.4 desaturate()}

\begin{verbatim}
void Fl_RGB_Image::desaturate ( ) [virtual]
The \texttt{desaturate()} method converts an image to grayscale. If the image contains an alpha channel (depth = 4), the alpha channel is preserved. An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
\end{verbatim}

Reimplemented from \texttt{Fl_Image}.

Reimplemented in \texttt{Fl_SVG_Image}.

\subsection*{12.116.3.5 draw()}

\begin{verbatim}
void Fl_RGB_Image::draw ( int X, int Y, int W, int H, int cx = 0, int cy = 0 ) [virtual]
Draws the image to the current drawing surface with a bounding box. Arguments \texttt{X,Y,W,H} specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the \texttt{cx} and \texttt{cy} arguments. In other words: \texttt{fl_push_clip(X,Y,W,H)} is applied, the image is drawn with its upper-left corner at \texttt{X-cx,Y-cy} and its own width and height, \texttt{fl_pop_clip()} is applied.
\end{verbatim}

Reimplemented from \texttt{Fl_Image}.

Reimplemented in \texttt{Fl_SVG_Image}.

\subsection*{12.116.3.6 label() [1/2]}

\begin{verbatim}
void Fl_RGB_Image::label ( Fl_Menu_Item * m ) [virtual]
This method is an obsolete way to set the image attribute of a menu item. \textbf{Deprecated} Please use \texttt{Fl_Menu_Item::image()} instead.
\end{verbatim}

Reimplemented from \texttt{Fl_Image}.

\texttt{Generated by Doxygen}
12.116.3.7 label() [2/2]

void Fl_RGB_Image::label ( 
    Fl_Widget ∗ widget ) [virtual]

This method is an obsolete way to set the image attribute of a widget or menu item.

**Deprecated** Please use Fl_Widget::image() or Fl_Widget::deimage() instead.

Reimplemented from Fl_Image.

12.116.3.8 max_size() [1/2]

static size_t Fl_RGB_Image::max_size ( ) [inline], [static]

Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

See also

    void Fl_RGB_Image::max_size(size_t)

12.116.3.9 max_size() [2/2]

static void Fl_RGB_Image::max_size ( 
    size_t size ) [inline], [static]

Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
The image size in bytes of an Fl_RGB_Image object is the value of the product w() ∗ h() ∗ d(). If this product exceeds size, the created object of a derived class of Fl_RGB_Image won’t be loaded with the image data. This does not apply to direct RGB image creation with Fl_RGB_Image::Fl_RGB_Image(const uchar ∗ bits, int W, int H, int D, int LD).
The default max_size() value is essentially infinite.

12.116.3.10 normalize()

virtual void Fl_RGB_Image::normalize ( ) [inline], [virtual]

Makes sure the object is fully initialized.
In particular, makes sure member variable array is non-null.
Reimplemented in Fl_SVG_Image.

12.116.3.11 uncache()

void Fl_RGB_Image::uncache ( ) [virtual]

If the image has been cached for display, delete the cache data.
This allows you to change the data used for the image and then redraw it without recreating an image object.
Reimplemented from Fl_Image.

12.116.4 Member Data Documentation

12.116.4.1 array

const uchar* Fl_RGB_Image::array

Points to the start of the object's data array.

See also

    class Fl_SVG_Image which delays initialization of this member variable.

The documentation for this class was generated from the following files:

- Fl_IMAGE.H
- Fl_IMAGE.cxx
The `Fl_Roller` widget is a "dolly" control commonly used to move 3D objects.

```cpp
#include <Fl_Roller.H>
```

Inheritance diagram for `Fl_Roller`:

```
Fl_Widget
|     |
|     |
Fl_Valuator
|     |
Fl_Roller
```

### Public Member Functions

- **`Fl_Roller (int X, int Y, int W, int H, const char ∗L=0)`**
  - Creates a new `Fl_Roller` widget using the given position, size, and label string.

- **`int handle (int)` FL_OVERRIDE**
  - Handles the specified event.

### Public Member Functions inherited from `Fl_Valuator`

- **`void bounds (double a, double b)`**
  - Sets the minimum (a) and maximum (b) values for the valuator widget.

- **`double clamp (double)`**
  - Clamps the passed value to the valuator range.

- **`virtual int format (char ∗)`**
  - Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.

- **`double increment (double, int)`**
  - Adds n times the step value to the passed value.

- **`double maximum () const`**
  - Gets the maximum value for the valuator.

- **`void maximum (double a)`**
  - Sets the maximum value for the valuator.

- **`double minimum () const`**
  - Gets the minimum value for the valuator.

- **`void minimum (double a)`**
  - Sets the minimum value for the valuator.

- **`void precision (int digits)`**
  - Sets the step value to $1.0 / 10^{digits}$.

- **`void range (double a, double b)`**
  - Sets the minimum and maximum values for the valuator.

- **`double round (double)`**
  - Round the passed value to the nearest step increment.

- **`double step () const`**
  - Gets or sets the step value.

- **`void step (double a, int b)`**
  - See `double Fl_Valuator::step() const`

- **`void step (double s)`**
  - See `double Fl_Valuator::step() const`.

- **`void step (int a)`**
See double Fl_Valuator::step() const

- double value () const
  Gets the floating point(double) value.
- int value (double)
  Sets the current value.

Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * as_gl_window () const
- virtual Fl_Group * as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * as_group () const
- virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * as_window () const
- void bind_deimage (Fl_Image * img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image * img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback * cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback * cb, Fl_Callback_User_Data * p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
  Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.
• void copy_label (const char *new_label)
  Sets the current label.
• void copy_tooltip (const char *text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
Sets the image to use as part of the widget label when in the inactive state.

- **void deimage(Fl_Image *img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound() const**
  Returns whether the inactive image is managed by the widget.

- **void do_callback(Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with default arguments.

- **void do_callback(Fl_Widget *widget, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void do_callback(Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void draw_label(int, int, int, int, Fl_Align) const**
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h() const**
  Gets the widget height.

- **virtual void hide()**
  Makes a widget invisible.

- **Fl_Image * image()**
  Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image * image() const**
  Gets the image that is used as part of the widget label when in the active state.

- **void image(Fl_Image &img)**
  Sets the image to use as part of the widget label when in the active state.

- **void image(Fl_Image *img)**
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound() const**
  Returns whether the image is managed by the widget.

- **int inside(const Fl_Widget *wgt) const**
  Checks if this widget is a child of wgt.

- **int is_label_copied() const**
  Returns whether the current label was assigned with copy_label().

- **const char * label() const**
  Gets the current label text.

- **void label(const char *text)**
  Sets the current label pointer.

- **void label(Fl_Labeltype a, const char *b)**
  Shortcut to set the label text and type in one call.

- **Fl_Color labelcolor() const**
  Gets the label color.

- **void labelcolor(Fl_Color c)**
  Sets the label color.

- **Fl_Font labelfont() const**
  Gets the font to use.

- **void labelfont(Fl_Font f)**
  Sets the font to use.

- **Fl_Fontsize labelsize() const**
  Gets the font size in pixels.

- **void labelsize(Fl_Fontsize pix)**
  Sets the font size in pixels.

- **Fl_Labeltype labeltype() const**
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takeevents () const
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.
• const char * tooltip () const
• void tooltip (const char *text)
  Sets the current tooltip text.

• Fl_Window *top_window () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window *top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const
  Gets the widget type.

• void type (uchar t)
  Sets the widget type.

• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void *user_data () const
  Gets the user data for this widget.

• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

• void user_data (void *v)
  Sets the user data for this widget.

• unsigned int visible () const
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

• void visible_focus (int v)
  Modifies keyboard focus navigation.

• int visible_r () const
  Returns whether a widget and all its parents are visible.

• int w () const
  Gets the widget width.

• Fl_When when () const
  Returns the conditions under which the callback is called.

• void when (uchar i)
  Sets the flags used to decide when a callback is called.

• Fl_Window *window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  Gets the widget position in its window.

• int y () const
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  Destroys the widget.

Protected Member Functions

• void draw () FL_OVERRIDE
  Draws the widget.
Protected Member Functions inherited from \texttt{Fl\_Valuator}

- \texttt{Fl\_Valuator} (int X, int Y, int W, int H, const char *L)
  
  Creates a new \texttt{Fl\_Valuator} widget using the given position, size, and label string.
- void \texttt{handle\_drag} (double newvalue)
  
  Called during a drag operation, after an FL\_WHEN\_CHANGED event is received and before the callback.
- void \texttt{handle\_push} ()
  
  Stores the current value in the previous value.
- void \texttt{handle\_release} ()
  
  Called after an FL\_WHEN\_RELEASE event is received and before the callback.
- int \texttt{horizontal} () const
  
  Tells if the valuator is an FL\_HORIZONTAL one.
- double \texttt{previous\_value} () const
  
  Gets the previous floating point value before an event changed it.
- void \texttt{set\_value} (double v)
  
  Sets the current floating point value.
- double \texttt{softclamp} (double)
  
  Clamps the value, but accepts \( v \) if the previous value is not already out of range.
- virtual void \texttt{value\_damage} ()
  
  Asks for partial redraw.

Protected Member Functions inherited from \texttt{Fl\_Widget}

- void \texttt{clear\_flag} (unsigned int c)
  
  Clears a flag in the flags mask.
- void \texttt{draw\_backdrop} () const
  
  If FL\_ALIGN\_IMAGE\_BACKDROP is set, the image or deimage will be drawn.
- void \texttt{draw\_box} () const
  
  Draws the widget box according its box style.
- void \texttt{draw\_box} (Fl\_Boxtype t, Fl\_Color c) const
  
  Draws a box of type \( t \), of color \( c \) at the widget's position and size.
- void \texttt{draw\_box} (Fl\_Boxtype t, int x, int y, int w, int h, Fl\_Color c) const
  
  Draws a box of type \( t \), of color \( c \) at the position \( X,Y \) and size \( W,H \).
- void \texttt{draw\_focus} () const
  
  Draws a focus rectangle around the widget.
- void \texttt{draw\_focus} (Fl\_Boxtype t, int X, int Y, int W, int H) const
  
  Draws a focus rectangle around the widget.
- void \texttt{draw\_focus} (Fl\_Boxtype t, int x, int y, int w, int h, Fl\_Color bg) const
  
  Draws a focus box for the widget at the given position and size.
- void \texttt{draw\_label} () const
  
  Draws the widget's label at the defined label position.
- void \texttt{draw\_label} (int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.
- \texttt{Fl\_Widget} (int x, int y, int w, int h, const char *=label*=0L)
  
  Creates a widget at the given position and size.
- unsigned int \texttt{flags} () const
  
  Gets the widget flags mask.
- void \texttt{h} (int v)
  
  \textit{Internal use only}.
- void \texttt{set\_flag} (unsigned int c)
  
  Sets a flag in the flags mask.
- void \texttt{w} (int v)
1322 Class Documentation

Internal use only.
• void x (int v)
  Internal use only.
• void y (int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget
• static void default_callback (Fl_Widget ∗widget, void ∗data)
  The default callback for all widgets that don't set a callback.
• static unsigned int label_shortcut (const char ∗t)
  Returns the Unicode value of the '&'x' shortcut in a given text.
• static int test_shortcut (const char ∗t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget
• enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

12.117.1 Detailed Description

The Fl_Roller widget is a "dolly" control commonly used to move 3D objects.
The roller can be controlled by clicking and dragging the mouse, by the corresponding arrow keys when the roller
has the keyboard focus, or by the mouse wheels when the mouse pointer is positioned over the roller widget.

![Fl_Roller](image)

Figure 12.38 Fl_Roller

12.117.2 Constructor & Destructor Documentation

12.117.2.1 Fl_Roller()

Fl_Roller::Fl_Roller ( int X,
  int Y,
int W,  
int H,  
const char ∗L = 0 )

Creates a new Fl_Roller widget using the given position, size, and label string.  
The default boxtype is FL_NO_BOX.  
Inherited destructor destroys the valuator.

12.117.3 Member Function Documentation

12.117.3.1 draw()    
void Fl_Roller::draw ( ) [protected], [virtual]  
Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as 
soon as possible, call redraw() instead.  
Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded 
scrollbar, you can do it (because draw() is virtual) like this:  
Fl_Widget ∗s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar  
s->draw(); // calls Fl_Scrollbar::draw()  

Implements Fl_Widget.

12.117.3.2 handle()  
int Fl_Roller::handle (    
    int event ) [virtual]

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-
circuit events that you don't handle. In this last case you should return the callee retval.  
One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class.

This requires knowledge of the details of the inherited class.  
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The 
effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in 
a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In 
such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by 
returning 1 even if you didn't do anything with it.

Parameters

| in | event  | the kind of event received |

Return values

| 0  | if the event was not used or understood |
| 1  | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Widget.

The documentation for this class was generated from the following files:

- Fl_Roller.H
- Fl_Roller.cxx
12.118  Fl_Round_Button Class Reference

Buttons generate callbacks when they are clicked by the user.

```
#include <Fl_Round_Button.h>
```

Inheritance diagram for Fl_Round_Button:

```
Fl_Widget
   |
   Fl_Button
   |
   Fl_Light_Button
   |
   Fl_Round_Button
   |
Fl_Radio_Round_Button
```

Public Member Functions

```
• Fl_Round_Button (int x, int y, int w, int h, const char *l=0)
  Creates a new Fl_Round_Button widget using the given position, size, and label string.
```

Public Member Functions inherited from Fl_Light_Button

```
• Fl_Light_Button (int x, int y, int w, int h, const char *l=0)
  Creates a new Fl_Light_Button widget using the given position, size, and label string.

• int handle (int) FL_OVERRIDE
  Handles the specified event.
```

Public Member Functions inherited from Fl_Button

```
• int clear ()
  Same as value(0).

• uchar compact ()
  Return true if buttons are rendered as compact buttons.

• void compact (uchar v)
  Decide if buttons should be rendered in compact mode.

• Fl_Boxtype down_box () const
  Returns the current down box type, which is drawn when value() is non-zero.

• void down_box (Fl_Boxtype b)
  Sets the down box type.

• Fl_Color down_color () const
  (for backwards compatibility)

• void down_color (unsigned c)
  (for backwards compatibility)

• Fl_Button (int X, int Y, int W, int H, const char *L=0)
  The constructor creates the button using the given position, size, and label.

• int set ()
  Same as value(1).

• void setonly ()
  Turns on this button and turns off all other radio buttons in the group (calling value(1) or set() does not do this).

• int shortcut () const
  ```
  ```
```
Returns the current shortcut key for the button.

* void **shortcut** (const char ∗s)
  (for backwards compatibility)
* void **shortcut** (int s)
  Sets the shortcut key to s.

* char **value** () const
  Returns the current value of the button (0 or 1).
* int **value** (int v)
  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

* void **_clear_fullscreen** ()
  * void **_set_fullscreen** ()
  * void **activate** ()
    Activates the widget.
* unsigned int **active** () const
  Returns whether the widget is active.
* int **active_r** () const
  Returns whether the widget and all of its parents are active.
* **Fl_Align** **align** () const
  Gets the label alignment.
  * void **align** (**Fl_Align** alignment)
    Sets the label alignment.
  * long **argument** () const
    Gets the current user data (long) argument that is passed to the callback function.
  * void **argument** (long v)
    Sets the current user data (long) argument that is passed to the callback function.
* virtual class **Fl_Gl_Window** ∗ **as_gl_window** ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
* virtual class **Fl_Gl_Window** const ∗ **as_gl_window** () const
* virtual class **Fl_Group** ∗ **as_group** ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
  * void **bind_deimage** (**Fl_Image** ∗img)
    Sets the image to use as part of the widget label when in the inactive state.
  * void **bind_deimage** (int f)
    Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
  * void **bind_image** (**Fl_Image** ∗img)
    Sets the image to use as part of the widget label when in the active state.
  * void **bind_image** (int f)
    Bind the image to the widget, so the widget will delete the image when it is no longer needed.
* **Fl_Boxtype** **box** () const
  Gets the box type of the widget.
  * void **box** (**Fl_Boxtype** new_box)
    Sets the box type for the widget.
* **Fl_Callback_p** **callback** () const
  Gets the current callback function for the widget.
• void callback (Fl_Callback *cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
  Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.
• void copy_label (const char *new_label)
  Sets the current label.
• void copy_tooltip (const char *text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image *deimage ()
Gets the image that is used as part of the widget label when in the inactive state.

- 

\begin{verbatim}
• const Fl_Image * deimage () const

Gets the image that is used as part of the widget label when in the inactive state.

• void deimage (Fl_Image &img)

Sets the image to use as part of the widget label when in the inactive state.

• void deimage (Fl_Image *img)

Sets the image to use as part of the widget label when in the inactive state.

• int deimage_bound () const

Returns whether the inactive image is managed by the widget.

• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

Calls the widget callback function with default arguments.

• void do_callback (Fl_Widget *wgt, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

Calls the widget callback function with arbitrary arguments.

• void do_callback (Fl_Widget *wgt, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

Calls the widget callback function with arbitrary arguments.

• void draw_label (int, int, int, int, Fl_Align) const

Draws the label in an arbitrary bounding box with an arbitrary alignment.

• int h () const

Gets the widget height.

• virtual void hide ()

Makes a widget invisible.

• Fl_Image * image ()

Gets the image that is used as part of the widget label when in the active state.

• const Fl_Image * image () const

Gets the image that is used as part of the widget label when in the active state.

• void image (Fl_Image &img)

Sets the image to use as part of the widget label when in the active state.

• void image (FlImageButton img)

Sets the image to use as part of the widget label when in the active state.

• int image_bound () const

Returns whether the image is managed by the widget.

• int inside (const Fl_Widget *wgt) const

Checks if this widget is a child of wgt.

• int is_label_copied () const

Returns whether the current label was assigned with copy_label().

• const char * label () const

Gets the current label text.

• void label (const char *text)

Sets the current label pointer.

• void label (Fl_Labeltype a, const char *b)

Shortcut to set the label text and type in one call.

• Fl_Color labelcolor () const

Gets the label color.

• void labelcolor (Fl_Color c)

Sets the label color.

• Fl_Font labelfont () const

Gets the font to use.

• void labelfont (Fl_Font f)

Sets the font to use.

• Fl_Fontsize labelsize () const

Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
   Sets the font size in pixels.

• Fl_Labeltype labeltype () const
   Gets the label type.

• void labeltype (Fl_Labeltype a)
   Sets the label type.

• void measure_label (int &ww, int &hh) const
   Sets width ww and height hh accordingly with the label size.

• bool needs_keyboard () const
   Returns whether this widget needs a keyboard.

• void needs_keyboard (bool needs)
   Sets whether this widget needs a keyboard.

• unsigned int output () const
   Returns if a widget is used for output only.

• Fl_Group * parent () const
   Returns a pointer to the parent widget.

• void parent (Fl_Group *p)
   Internal use only - "for hacks only".

• void position (int X, int Y)
   Repositions the window or widget.

• void redraw ()
   Schedules the drawing of the widget.

• void redraw_label ()
   Schedules the drawing of the label.

• virtual void resize (int x, int y, int w, int h)
   Changes the size or position of the widget.

• Fl_Color selection_color () const
   Gets the selection color.

• void selection_color (Fl_Color a)
   Sets the selection color.

• void set_active ()
   Marks the widget as active without sending events or changing focus.

• void set_changed ()
   Marks the value of the widget as changed.

• void set_output ()
   Sets a widget to output only.

• void set_visible ()
   Makes the widget visible.

• void set_visible_focus ()
   Enables keyboard focus navigation with this widget.

• int shortcut_label () const
   Returns whether the widget's label uses '&' to indicate shortcuts.

• void shortcut_label (int value)
   Sets whether the widget's label uses '&' to indicate shortcuts.

• virtual void show ()
   Makes a widget visible.

• void size (int W, int H)
   Changes the size of the widget.

• int take_focus ()
   Gives the widget the keyboard focus.

• unsigned int takesevents () const

Generated by Doxygen
Returns if the widget is able to take events.

- `int test_shortcut()`
  Returns true if the widget's label contains the entered '&x' shortcut.

- `const char * tooltip()`
  Gets the current tooltip text.

- `void tooltip(const char *text)`
  Sets the current tooltip text.

- `Fl_Window * top_window()`
  Returns a pointer to the top-level window for the widget.

- `Fl_Window * top_window_offset(int &xoff, int &yoff)`
  Finds the x/y offset of the current widget relative to the top-level window.

- `uchar type()`
  Gets the widget type.

- `void type(uchar t)`
  Sets the widget type.

- `int use_accents_menu()`
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- `void * user_data()`
  Gets the user data for this widget.

- `void user_data(Fl_Callback_User_Data *v, bool auto_free)`
  Sets the user data for this widget.

- `void user_data(void *v)`
  Sets the user data for this widget.

- `unsigned int visible()`
  Returns whether a widget is visible.

- `unsigned int visible_focus()`
  Checks whether this widget has a visible focus.

- `void visible_focus(int v)`
  Modifies keyboard focus navigation.

- `int visible_r()`
  Returns whether a widget and all its parents are visible.

- `int w()`
  Gets the widget width.

- `Fl_When when()`
  Returns the conditions under which the callback is called.

- `void when(uchar i)`
  Sets the flags used to decide when a callback is called.

- `Fl_Window * window()`
  Returns a pointer to the nearest parent window up the widget hierarchy.

- `int x()`
  Gets the widget position in its window.

- `int y()`
  Gets the widget position in its window.

- `virtual ~Fl_Widget()`
  Destroys the widget.
Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.

• static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.

• static int test_shortcut (const char *t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

• enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31}
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Light_Button

• void draw () FL_OVERRIDE
  Draws the widget.

Protected Member Functions inherited from Fl_Button

• void simulate_key_action ()

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.

• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• void draw_box () const
  Draws the widget box according its box style.

• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

• void draw_focus () const
  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
12.118 Fl_Round_Button Class Reference

- **void draw_label()** const
  
  Draws the widget's label at the defined label position.

- **void draw_label(int, int, int, int)** const
  
  Draws the label in an arbitrary bounding box.

- **Fl_Widget(int x, int y, int w, int h, const char *label=0L)**
  
  Creates a widget at the given position and size.

- **unsigned int flags()** const
  
  Gets the widget flags mask.

- **void h(int v)**
  
  Internal use only.

- **void set_flag(unsigned int c)**
  
  Sets a flag in the flags mask.

- **void w(int v)**
  
  Internal use only.

- **void x(int v)**
  
  Internal use only.

- **void y(int v)**
  
  Internal use only.

**Static Protected Member Functions inherited from Fl_Button**

- **static void key_release_timeout(void *)**

**Static Protected Attributes inherited from Fl_Button**

- **static Fl_Widget_Tracker* key_release_tracker = 0**

12.118.1 Detailed Description

Buttons generate callbacks when they are clicked by the user. You control exactly when and how by changing the values for **type()** and **when()**.

The **Fl_Round_Button** subclass displays the "on" state by turning on a light, rather than drawing pushed in. The shape of the "light" is initially set to **FL_ROUND_DOWN_BOX**. The color of the light when on is controlled with **selection_color()**, which defaults to **FL_FOREGROUND_COLOR**.

12.118.2 Constructor & Destructor Documentation

12.118.2.1 **Fl_Round_Button()**

**Fl_Round_Button::Fl_Round_Button (**

  int X,
  int Y,
  int W,
  int H,
  const char * L = 0)

Creates a new **Fl_Round_Button** widget using the given position, size, and label string.
The Fl_Round_Button subclass displays the "ON" state by turning on a light, rather than drawing pushed in. The default box type is FL_NO_BOX, which draws the label w/o a box right of the checkmark. The shape of the "light" is set with down_box() and its default value is FL_ROUND_DOWN_BOX. The color of the light when on is controlled with selection_color(), which defaults to FL_FOREGROUND_COLOR (usually black).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X,Y,W,H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Round_Button.H
- Fl_Round_Button.cxx

### 12.119 Fl_Round_Clock Class Reference

A clock widget of type FL_ROUND_CLOCK.

```c
#include <Fl_Round_Clock.H>
```

Inheritance diagram for Fl_Round_Clock:

```
Fl_Widget
 |   |
|   | Fl_Clock_Output
|   |
|   | Fl_Clock
|   |
|   | Fl_Round_Clock
```

**Public Member Functions**

- `Fl_Round_Clock (int X, int Y, int W, int H, const char *L=0)`
  
  Creates the clock widget, setting his type and box.

**Public Member Functions inherited from Fl_Clock**

- `Fl_Clock (int X, int Y, int W, int H, const char *L=0)`
  
  Create an Fl_Clock widget using the given position, size, and label string.

- `Fl_Clock (uchar t, int X, int Y, int W, int H, const char *L)`
  
  Create an Fl_Clock widget using the given clock type t, position, size, and label string.

- `int handle (int) FL_OVERRIDE`
  
  Handles the specified event.

- `~Fl_Clock ()`
  
  The destructor removes the clock.

**Public Member Functions inherited from Fl_Clock_Output**

- `Fl_Clock_Output (int X, int Y, int W, int H, const char *L=0)`
  
  Create a new Fl_Clock_Output widget with the given position, size and label.

- `int hour () const`
  
  Returns the displayed hour (0 to 23).
12.119 Fl_Round_Clock Class Reference

- **int minute () const**
  Returns the displayed minute (0 to 59).

- **int second () const**
  Returns the displayed second (0 to 60, 60=leap second).

- **int shadow () const**
  Returns the shadow drawing mode of the hands.

- **void shadow (int mode)**
  Sets the shadow drawing mode of the hands.

- **ulong value () const**
  Returns the displayed time.

- **void value (int H, int m, int s)**
  Set the displayed time.

- **void value (ulong v)**
  Set the displayed time.

Public Member Functions inherited from Fl_Widget

- **void _clear_fullscreen ()**
- **void _set_fullscreen ()**
- **void activate ()**
  Activates the widget.

- **unsigned int active () const**
  Returns whether the widget is active.

- **int active_r () const**
  Returns whether the widget and all of its parents are active.

- **Fl_Align align () const**
  Gets the label alignment.

- **void align (Fl_Align alignment)**
  Sets the label alignment.

- **long argument () const**
  Gets the current user data (long) argument that is passed to the callback function.

- **void argument (long v)**
  Sets the current user data (long) argument that is passed to the callback function.

- **virtual class Fl_Gl_Window * as_gl_window ()**
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- **virtual class Fl_Gl_Window const * as_gl_window () const**

- **virtual Fl_Group * as_group ()**
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- **virtual Fl_Group const * as_group () const**

- **virtual Fl_Window * as_window ()**
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- **virtual Fl_Window const * as_window () const**

- **void bind_deimage (Fl_Image * img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **void bind_deimage (int f)**
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- **void bind_image (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the active state.

- **void bind_image (int f)**
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype box () const**
Gets the box type of the widget.

- **void** box (**Fl_Boxtype** new_box)
  
  Sets the box type for the widget.

- **void** callback (**Fl_Callback** *cb)
  
  Sets the current callback function for the widget.

- **void** callback (**Fl_Callback** *cb, **Fl_Callback_User_Data** *p, bool auto_free)
  
  Sets the current callback function and managed user data for the widget.

- **void** callback (**Fl_Callback** *cb, **void** *p)
  
  Sets the current callback function and data for the widget.

- **void** callback (**Fl_Callback0** *cb)
  
  Sets the current callback function for the widget.

- **void** callback (**Fl_Callback1** *cb, long p=0)
  
  Sets the current callback function for the widget.

- **unsigned int** changed ()
  
  Checks if the widget value changed since the last callback.

- **void** clear_active ()
  
  Marks the widget as inactive without sending events or changing focus.

- **void** clear_changed ()
  
  Marks the value of the widget as unchanged.

- **void** clear_damage (uchar c=0)
  
  Clears or sets the damage flags.

- **void** clear_output ()
  
  Sets a widget to accept input.

- **void** clear_visible ()
  
  Hides the widget.

- **void** clear_visible_focus ()
  
  Disables keyboard focus navigation with this widget.

- **Fl_Color** color ()
  
  Gets the background color of the widget.

- **void** color (**Fl_Color** bg)
  
  Sets the background color of the widget.

- **void** color (**Fl_Color** bg, **Fl_Color** sel)
  
  Sets the background and selection color of the widget.

- **Fl_Color** color2 ()
  
  For back compatibility only.

- **void** color2 (unsigned a)
  
  For back compatibility only.

- **int** contains (const **Fl_Widget** *w) const
  
  Checks if w is a child of this widget.

- **void** copy_label (const char *new_label)
  
  Sets the current label.

- **void** copy_tooltip (const char *text)
  
  Sets the current tooltip text.

- **uchar** damage ()
  
  Returns non-zero if draw() needs to be called.

- **void** damage (uchar c)
  
  Sets the damage bits for the widget.

- **void** damage (uchar c, int x, int y, int w, int h)
  
  Sets the damage bits for an area inside the widget.
• int **damage_resize** (int, int, int, int)
  
  Internal use only.

• void **deactivate** ()
  
  Deactivates the widget.

• Fl_Image **deimage** ()
  
  Gets the image that is used as part of the widget label when in the inactive state.

• const Fl_Image **deimage** () const
  
  Gets the image that is used as part of the widget label when in the inactive state.

• void **deimage** (Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the inactive state.

• void **deimage** (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the inactive state.

• int **deimage_bound** () const
  
  Returns whether the inactive image is managed by the widget.

• void **do_callback** (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with default arguments.

• void **do_callback** (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with arbitrary arguments.

• void **do_callback** (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with arbitrary arguments.

• void **draw_label** (int, int, int, int, Fl_Align) const
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• int **h** () const
  
  Gets the widget height.

• virtual void **hide** ()
  
  Makes a widget invisible.

• Fl_Image **image** ()
  
  Gets the image that is used as part of the widget label when in the active state.

• const Fl_Image **image** () const
  
  Gets the image that is used as part of the widget label when in the active state.

• void **image** (Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the active state.

• void **image** (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the active state.

• int **image_bound** () const
  
  Returns whether the image is managed by the widget.

• int **inside** (const Fl_Widget *wgt) const
  
  Checks if this widget is a child of wgt.

• int **is_label_copied** () const
  
  Returns whether the current label was assigned with copy_label().

• const char **label** () const
  
  Gets the current label text.

• void **label** (const char *text)
  
  Sets the current label pointer.

• void **label** (Fl_Labeltype a, const char *b)
  
  Shortcut to set the label text and type in one call.

• Fl_Color **labelcolor** () const
  
  Gets the label color.

• void **labelcolor** (Fl_Color c)
  
  Sets the label color.

• Fl_Font **labelfont** () const
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void \texttt{size} (int W, int H)
  \textit{Changes the size of the widget.}

• int \texttt{take\_focus} ()
  \textit{Gives the widget the keyboard focus.}

• unsigned int \texttt{take\_events} () const
  \textit{Returns if the widget is able to take events.}

• int \texttt{test\_shortcut} ()
  \textit{Returns true if the widget's label contains the entered 'x' shortcut.}

• const char \* \texttt{tooltip} () const
  \textit{Gets the current tooltip text.}

• void \texttt{tooltip} (const char \*text)
  \textit{Sets the current tooltip text.}

  \texttt{FI\_Window} \* \texttt{top\_window} () const
  \textit{Returns a pointer to the top-level window for the widget.}

  \texttt{FI\_Window} \* \texttt{top\_window\_offset} (int \&xoff, int \&yoff) const
  \textit{Finds the x/y offset of the current widget relative to the top-level window.}

• uchar \texttt{type} () const
  \textit{Gets the widget type.}

• void \texttt{type} (uchar t)
  \textit{Sets the widget type.}

• int \texttt{use\_accents\_menu} ()
  \textit{Returns non zero if MAC\_USE\_ACCENTS\_MENU flag is set, 0 otherwise.}

• void \texttt{user\_data} () const
  \textit{Gets the user data for this widget.}

• void \texttt{user\_data} (FI\_Callback\_User\_Data \*v, bool auto\_free)
  \textit{Sets the user data for this widget.}

• void \texttt{user\_data} (void \*v)
  \textit{Sets the user data for this widget.}

• unsigned int \texttt{visible} () const
  \textit{Returns whether a widget is visible.}

• unsigned int \texttt{visible\_focus} () const
  \textit{Checks whether this widget has a visible focus.}

• void \texttt{visible\_focus} (int v)
  \textit{Modifies keyboard focus navigation.}

• int \texttt{visible\_r} () const
  \textit{Returns whether a widget and all its parents are visible.}

• int \texttt{w} () const
  \textit{Gets the widget width.}

• \texttt{FI\_When\_when} () const
  \textit{Returns the conditions under which the callback is called.}

• void \texttt{when} (uchar i)
  \textit{Sets the flags used to decide when a callback is called.}

• \texttt{FI\_Window} \* \texttt{window} () const
  \textit{Returns a pointer to the nearest parent window up the widget hierarchy.}

• int \texttt{x} () const
  \textit{Gets the widget position in its window.}

• int \texttt{y} () const
  \textit{Gets the widget position in its window.}

• virtual \texttt{~FI\_Widget} ()
  \textit{Destroys the widget.}
### Additional Inherited Members

#### Static Public Member Functions inherited from Fl_Widget

- `static void default_callback (Fl_Widget *widget, void *data)`
  *The default callback for all widgets that don’t set a callback.*

- `static unsigned int label_shortcut (const char *t)`
  *Returns the Unicode value of the ‘&x’ shortcut in a given text.*

- `static int test_shortcut (const char *t, const bool require_alt=false)`
  *Returns true if the given text t contains the entered ‘&x’ shortcut.*

#### Protected Types inherited from Fl_Widget

- `enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  , OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  , MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  , GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 , NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 , MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

#### Protected Member Functions inherited from Fl_Clock_Output

- `void draw () FL_OVERRIDE`
  *Draw clock with current position and size.*

- `void draw (int X, int Y, int W, int H)`
  *Draw clock with the given position and size.*

#### Protected Member Functions inherited from Fl_Widget

- `void clear_flag (unsigned int c)`
  *Clears a flag in the flags mask.*

- `void draw_backdrop () const`
  *If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.*

- `void draw_box () const`
  *Draws the widget box according its box style.*

- `void draw_box (Fl_Boxtype t, Fl_Color c) const`
  *Draws a box of type t, of color c at the widget’s position and size.*

- `void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const`
  *Draws a box of type t, of color c at the position X,Y and size W,H.*

- `void draw_focus () const`
  *Draws a focus rectangle around the widget.*

- `void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const`
  *Draws a focus rectangle around the widget.*

- `void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const`
  *Draws a focus box for the widget at the given position and size.*

- `void draw_label () const`
  *Draws the widget’s label at the defined label position.*
12.120 Fl_Scheme Class Reference

- void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.
- unsigned int flags () const
  Gets the widget flags mask.
- void h (int v)
  Internal use only.
- void set_flag (unsigned int c)
  Sets a flag in the flags mask.
- void w (int v)
  Internal use only.
- void x (int v)
  Internal use only.
- void y (int v)
  Internal use only.

12.119.1 Detailed Description

A clock widget of type FL_ROUND_CLOCK.
Has no box.

12.119.2 Constructor & Destructor Documentation

12.119.2.1 Fl_Round_Clock()

Fl_Round_Clock::Fl_Round_Clock (int X, int Y, int W, int H, const char *L = 0L)

Creates the clock widget, setting his type and box.
Create an Fl_Round_Clock widget using the given position, size, and label string.
The clock type is FL_ROUND_CLOCK and the boxtype is FL_NO_BOX.
This constructor is the same as Fl_Clock(FL_ROUND_CLOCK, X, Y, W, H, L).

See also

Fl_Clock(uchar, int, int, int, int, const char *)

Parameters

| in | X,Y,W,H | position and size of the widget |
| in | L | widget label, default is no label |

The documentation for this class was generated from the following files:

- Fl_Round_Clock.H
- Fl_Clock.cxx

12.120 Fl_Scheme Class Reference

Static Public Member Functions

- static int add_scheme_name (const char *name)
Add a scheme name to the list of known schemes.

• static const char ** names ()

  Return a list of all known scheme names.

• static int num_schemes ()

  Return the number of currently registered schemes.

### 12.120.1 Member Function Documentation

#### 12.120.1.1 add_scheme_name()

```cpp
int Fl_Scheme::add_scheme_name (  
  const char * name ) [static]
```

Add a scheme name to the list of known schemes.

This method is public in FLTK 1.4.0 because derived classes of `Fl_Scheme` are not yet implemented. Thus, users implementing their own schemes can use this method to add the scheme name to the list of known schemes which is for instance used in `Fl_Scheme::names()`.

**Note**

**Attention!** In a future version, when subclasses of `Fl_Scheme` will be implemented, this method will either be replaced by another protected method or it will no longer do anything (kept only for ABI reasons).

The new scheme name must consist of valid ASCII characters as described below:

- lowercase letters 'a'-'z'
- numbers '0'-'9'
- any character in "$+_." (w/o the quotes).

The name must not be longer than 12 ASCII characters (bytes). The new scheme name is added to the end of the unordered list.

**Note**

Call this method only once for each scheme name. If the returned value is \(\leq 0\) you should check the scheme name.

The given scheme name is copied and may be freed directly after the call to `add_scheme_name()`.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>New scheme name</th>
</tr>
</thead>
</table>

**Returns**

The new number of schemes if the name was successfully added. This is the same as the index of the scheme + 1.

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>Scheme name already exists</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Invalid character(s) in name</td>
</tr>
<tr>
<td>-2</td>
<td>The name is too long</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Since 1.4.0

12.120.1.2 names()

const char ** Fl_Scheme::names ( ) [static]

Return a list of all known scheme names. This list is only valid until a new scheme is added or one is removed. It is possible that scheme names are appended to the list during the runtime of the program but schemes can’t be removed.

Getting the list of known schemes can be useful to populate a menu of scheme choices to let the user select a scheme. You should process the names immediately and you should never store a pointer to the list or any individual name for later reference because the location of the list can be changed (reallocated) when schemes are added.

The list of scheme names is null-terminated.

Note

Currently (in FLTK 1.4.0) schemes can only be added to the list and not removed from the list. This may change in a later version.

Returns

List of currently known scheme names.

12.120.1.3 num_schemes()

static int Fl_Scheme::num_schemes ( ) [inline], [static]

Return the number of currently registered schemes.

Returns

Number of registered schemes.

The documentation for this class was generated from the following files:

• Fl_Scheme.H
• Fl_Scheme.cxx

12.121 Fl_Scheme_Choice Class Reference

Inheritance diagram for Fl_Scheme_Choice:

Public Member Functions

• Fl_Scheme_Choice (int X, int Y, int W, int H, const char *L=0)
  
The constructor initializes the Fl_Scheme_Choice object with all known schemes.

• int handle (int event) FL_OVERRIDE
  
Handle FLTK events.

• virtual void init_value ()
  
Public method to initialize the value of the Fl_Scheme_Choice widget.
Public Member Functions inherited from Fl_Choice

- **Fl_Choice** (int X, int Y, int W, int H, const char ∗L=0)
  
  Create a new Fl_Choice widget using the given position, size and label string.

- int **value** () const
  
  Gets the index of the last item chosen by the user.

- int **value** (const Fl_Menu_Item ∗v)
  
  Sets the currently selected value using a pointer to menu item.

- int **value** (int v)
  
  Sets the currently selected value using the index into the menu item array.

Public Member Functions inherited from Fl_Menu_

- int **add** (const char ∗)
  
  This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with ↑ separating the menu items, and tab separating the menu item names from an optional shortcut string.

- int **add** (const char ∗, int shortcut, Fl_Callback ∗c, void ∗d=0, int e=0)
  
  Adds a new menu item.

- void **clear** ()
  
  Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.

- int **clear_submenu** (int index)
  
  Clears the specified submenu pointed to by index of all menu items.

- void **copy** (const Fl_Menu_Item ∗m, void ∗user_data=0)
  
  Sets the menu array pointer with a copy of m that will be automatically deleted.

- Fl_Boxtype **down_box** () const
  
  This box type is used to surround the currently-selected items in the menus.

- void **down_box** (Fl_Boxtype b)
  
  Sets the box type used to surround the currently-selected items in the menus.

- Fl_Color **down_color** () const
  
  For back compatibility, same as selection_color()

- void **down_color** (unsigned c)
  
  For back compatibility, same as selection_color()

- int **find_index** (const char ∗name) const
  
  Find the menu item index for a given menu pathname, such as "Edit/Copy".

- int **find_index** (const Fl_Menu_Item ∗item) const
  
  Find the index into the menu array for a given item.

- int **find_index** (Fl_Callback ∗cb) const
  
  Find the index into the menu array for a given callback cb.

- const Fl_Menu_Item ∗**find_item** (const char ∗name)
  
  Find the menu item for a given menu pathname, such as "Edit/Copy".

- const Fl_Menu_Item ∗**find_item** (Fl_Callback ∗cb)
  
  Find the menu item for the given callback cb.

- const Fl_Menu_Item ∗**find_item_with_argument** (long v)
  
  Find the menu item for the given user argument v.

- const Fl_Menu_Item ∗**find_item_with_user_data** (void ∗)
  
  Find the menu item for the given user data v.

- Fl_Menu_ (int, int, int, int, const char ∗L=0)
  
  Creates a new Fl_Menu_ widget using the given position, size, and label string.

- void **global** ()
  
  Make the shortcuts for this menu work no matter what window has the focus when you type it.
• int insert (int index, const char *, int shortcut, Fl_Callback *, void *=0, int e=0)
  Inserts a new menu item at the specified index position.

• int insert (int index, const char *a, const char *b, Fl_Callback *c, void *=0, int e=0)
  See int Fl_Menu_::insert(const char* label, int shortcut, Fl_Callback*, void *=user_data=0, int flags=0)

• int item_pathname (char *name, int namelen, const Fl_Menu_Item *finditem=0) const
  Get the menu 'pathname' for the specified menuitem.

• const Fl_Menu_Item * menu () const
  Returns a pointer to the array of Fl_Menu_Items.

• void menu (const Fl_Menu_Item *m)
  Sets the menu array pointer directly.

• Fl_Boxtype menu_box () const
  Get the box type for the menu popup windows.

• void menu_box (Fl_Boxtype b)
  Set the box type for the menu popup windows.

• const Fl_Menu_Item * menu_end ()
  Finishes menu modifications and returns menu().

• int mode (int i) const
  Gets the flags of item i.

• void mode (int i, int fl)
  Sets the flags of item i.

• const Fl_Menu_Item * mvalue () const
  Returns a pointer to the last menu item that was picked.

• const Fl_Menu_Item * picked (const Fl_Menu_Item *)
  When user picks a menu item, call this.

• const Fl_Menu_Item * prev_mvalue () const
  Returns a pointer to the menu item that was picked before the current one was picked.

• void remove (int)
  Deletes item i from the menu.

• void replace (int, const char *)
  Changes the text of item i.

• void setonly (Fl_Menu_Item *item)
  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.

• void shortcut (int i, int s)
  Changes the shortcut of item i to s.

• int size () const
  This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.

• void size (int W, int H)
  Returns the menu item with the entered shortcut (key value).

• const char * text () const
  Returns the title of the last item chosen.

• const char * text (int i) const
  Returns the title of item i.

• Fl_Color textcolor () const
  Get the current color of menu item labels.

• void textcolor (Fl_Color c)
  Sets the current color of menu item labels.

• Fl_Font textfont () const
  Gets the current font of menu item labels.

• void textfont (Fl_Font c)
  Sets the current font of menu item labels.
• **Fl_Fontsize** **textsize** () const
  
  Gets the font size of menu item labels.

• void **textsize** (Fl_Fontsize c)

  Sets the font size of menu item labels.

• int **value** () const

  Returns the index into menu() of the last item chosen by the user.

• int **value** (const Fl_Menu_Item *)

  The value is the index into menu() of the last item chosen by the user.

• int **value** (int i)

  The value is the index into menu() of the last item chosen by the user.

### Public Member Functions inherited from Fl_Widget

• void **_clear_fullscreen** ()

• void **_set_fullscreen** ()

  Activates the widget.

• unsigned int **active** () const

  Returns whether the widget is active.

• int **active_r** () const

  Returns whether the widget and all of its parents are active.

• Fl_Align **align** () const

  Gets the label alignment.

• void **align** (Fl_Align alignment)

  Sets the label alignment.

• long **argument** () const

  Gets the current user data (long) argument that is passed to the callback function.

• void **argument** (long v)

  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window * **as_gl_window** ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const * **as_gl_window** () const

• virtual Fl_Group * **as_group** ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Group const * **as_group** () const

• virtual Fl_Window * **as_window** ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window const * **as_window** () const

• void **bind_deimage** (Fl_Image * img)

  Sets the image to use as part of the widget label when in the inactive state.

• void **bind_deimage** (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

• void **bind_image** (Fl_Image * img)

  Sets the image to use as part of the widget label when in the active state.

• void **bind_image** (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

• Fl_Boxtype **box** () const

  Gets the box type of the widget.

• void **box** (Fl_Boxtype new_box)

  Sets the box type for the widget.

• Fl_Callback **callback** () const
Gets the current callback function for the widget.
• void callback (Fl_Callback ∗cb)
  Sets the current callback function for the widget.

Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

Sets the current callback function for the widget.
• void callback (Fl_Callback ∗cb, void ∗p)

Sets the current callback function for the widget.
• void callback (Fl_Callback0 ∗cb)

Sets the current callback function for the widget.
• void callback (Fl_Callback1 ∗cb, long p=0)

Checks if the widget value changed since the last callback.
• unsigned int changed () const
  Marks the widget as inactive without sending events or changing focus.
• void clear_active ()

Marks the value of the widget as unchanged.
• void clear_changed ()
  Clears or sets the damage flags.
• void clear_damage (uchar c=0)
  Sets a widget to accept input.
• void clear_input ()

Hides the widget.
• void clear_visible ()
  Disables keyboard focus navigation with this widget.
• void clear_visible_focus ()
• Fl_Color color () const
  Gets the background color of the widget.
• Fl_Color color (Fl_Color bg)
  Sets the background color of the widget.
• Fl_Color color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget ∗w) const
  Checks if w is a child of this widget.
• void copy_label (const char ∗new_label)
  Sets the current label.
• void copy_tooltip (const char ∗text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• **FL_Image * deimage ()**
  
  Gets the image that is used as part of the widget label when in the inactive state.

• **const FL_Image * deimage () const**

  Gets the image that is used as part of the widget label when in the inactive state.

• **void deimage (FL_Image &img)**

  Sets the image to use as part of the widget label when in the inactive state.

• **void deimage (FL_Image *img)**

  Sets the image to use as part of the widget label when in the inactive state.

• **int deimage_bound () const**

  Returns whether the inactive image is managed by the widget.

• **void do_callback (FL_Callback_Reason reason=FL_REASON_UNKNOWN)**

  Calls the widget callback function with default arguments.

• **void do_callback (FL_Widget *widget, long arg, FL_Callback_Reason reason=FL_REASON_UNKNOWN)**

  Calls the widget callback function with arbitrary arguments.

• **void do_callback (FL_Widget *widget, void *arg=0, FL_Callback_Reason reason=FL_REASON_UNKNOWN)**

  Calls the widget callback function with arbitrary arguments.

• **void draw_label (int, int, int, int, Fl_Align) const**

  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• **int h () const**

  Gets the widget height.

• **virtual void hide ()**

  Makes a widget invisible.

• **FL_Image * image ()**

  Gets the image that is used as part of the widget label when in the active state.

• **const FL_Image * image () const**

  Gets the image that is used as part of the widget label when in the active state.

• **void image (FL_Image &img)**

  Sets the image to use as part of the widget label when in the active state.

• **void image (FL_Image *img)**

  Sets the image to use as part of the widget label when in the active state.

• **int image_bound () const**

  Returns whether the image is managed by the widget.

• **int inside (const FL_Widget *wgt) const**

  Checks if this widget is a child of wgt.

• **int is_label_copied () const**

  Returns whether the current label was assigned with copy_label().

• **const char * label () const**

  Gets the current label text.

• **void label (const char *text)**

  Sets the current label pointer.

• **void label (FL_Labeltype a, const char *b)**

  Shortcut to set the label text and type in one call.

• **FL_Color labelcolor () const**

  Gets the label color.

• **void labelcolor (FL_Color c)**

  Sets the label color.

• **FL_Font labelfont () const**

  Gets the font to use.

• **void labelfont (FL_Font f)**

  Sets the font to use.

• **FL_Fontsize labelsize () const**
Gets the font size in pixels.

- void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.

- Fl_Labeltype labeltype () const
  Gets the label type.

- void labeltype (Fl_Labeltype a)
  Sets the label type.

- void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.

- bool needs_keyboard () const
  Returns whether this widget needs a keyboard.

- void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.

- unsigned int output () const
  Returns if a widget is used for output only.

- Fl_Group * parent () const
  Returns a pointer to the parent widget.

- void parent (Fl_Group *p)
  Internal use only - “for hacks only”.

- void position (int X, int Y)
  Repositions the window or widget.

- void redraw ()
  Schedules the drawing of the widget.

- void redraw_label ()
  Schedules the drawing of the label.

- virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.

- Fl_Color selection_color () const
  Gets the selection color.

- void selection_color (Fl_Color a)
  Sets the selection color.

- void set_active ()
  Marks the widget as active without sending events or changing focus.

- void set_changed ()
  Marks the value of the widget as changed.

- void set_output ()
  Sets a widget to output only.

- void set_visible ()
  Makes the widget visible.

- void set_visible_focus ()
  Enables keyboard focus navigation with this widget.

- int shortcut_label () const
  Returns whether the widget's label uses '&’ to indicate shortcuts.

- void shortcut_label (int value)
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.

- virtual void show ()
  Makes a widget visible.

- void size (int W, int H)
  Changes the size of the widget.

- int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takeevents () const
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.
• const char * tooltip () const
  Gets the current tooltip text.
• void tooltip (const char *text)
  Sets the current tooltip text.
• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.

Static Protected Member Functions
• static void scheme_cb_ (Fl_Widget *w, void *)
  Internal Fl_Scheme_Choice callback function (protected).
Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void **default_callback**( Fl_Widget *widget, void *data)  
  The default callback for all widgets that don't set a callback.
- static unsigned int **labelShortcut**( const char *t)  
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int **testShortcut**( const char *t, const bool require_alt=false)  
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,  
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,  
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPY_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,  
  MENU_WINDOW = 1<<12, TOOLBOX_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,  
  GROUP_RELATIVE = 1<<16, COPY_TOOTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,  
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,  
  AUTO_DELETE_USER_DATA = 1<<23,  
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31  
} flags possible values enumeration.

Protected Member Functions inherited from Fl_Choice

- void **draw** FL_OVERRIDE  
  Draws the widget.

Protected Member Functions inherited from Fl_Menu_

- int **itemGetName**( char *name, int namelen, const Fl_Menu_Item *findItem, const Fl_Menu_Item *menu=0) const

Protected Member Functions inherited from Fl_Widget

- void **clearFlag**( unsigned int c)  
  Clears a flag in the flags mask.
- void **drawBackdrop**() const  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **drawBox**() const  
  Draws the widget box according its box style.
- void **drawBox**( Fl_Boxtype t, Fl_Color c) const  
  Draws a box of type t, of color c at the widget’s position and size.
- void **drawBox**( Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const  
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void **drawFocus**() const  
  Draws a focus rectangle around the widget.
- void **drawFocus**( Fl_Boxtype t, int X, int Y, int W, int H) const  
  Draws a focus rectangle around the widget.
- void **drawFocus**( Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
Draws a focus box for the widget at the given position and size.

- void `draw_label()` const
  Draws the widget's label at the defined label position.

- void `draw_label(int, int, int, int)` const
  Draws the label in an arbitrary bounding box.

- `Fl_Widget(int x, int y, int w, int h, const char *label=0L)`
  Creates a widget at the given position and size.

- unsigned int `flags()` const
  Gets the widget flags mask.

- void `h(int v)`
  Internal use only.

- void `set_flag(unsigned int c)`
  Sets a flag in the flags mask.

- void `w(int v)`
  Internal use only.

- void `x(int v)`
  Internal use only.

- void `y(int v)`
  Internal use only.

Protected Attributes inherited from `Fl_Menu`

- uchar `alloc`
- uchar `down_box_`
- `Fl_Boxtype menu_box_`
- `Fl_Color textcolor_`
- `Fl_Font textfont_`
- `Fl_Fontsize textsize_`

12.121.1 Constructor & Destructor Documentation

12.121.1.1 `Fl_Scheme_Choice()`

`Fl_Scheme_Choice::Fl_Scheme_Choice(int X, int Y, int W, int H, const char *L=0L)`
The constructor initializes the `Fl_Scheme_Choice` object with all known schemes.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int <code>X,Y</code></td>
<td>Widget coordinates</td>
</tr>
<tr>
<td>int <code>W,H</code></td>
<td>Widget size (width, height)</td>
</tr>
<tr>
<td><code>L</code></td>
<td>Widget label (default: NULL, no label)</td>
</tr>
</tbody>
</table>

12.121.2 Member Function Documentation

12.121.2.1 `handle()`

`int Fl_Scheme_Choice::handle(int event)` [virtual]
Handle FLTK events.
This widget uses FL_SHOW and some other events to initialize its `value()` according to the current scheme. All events are also handled by the base class `Fl_Choice`.

**Parameters**

| in | event |

**Returns**

1 if the event was used, 0 otherwise

Reimplemented from `Fl_Choice`.

### 12.121.2.2 init_value()

```cpp
void Fl_Scheme_Choice::init_value ( ) [virtual]
```

Public method to initialize the value of the `Fl_Scheme_Choice` widget. Normally you don't need to call this unless you change the current scheme by calling `Fl::scheme(const char *)`. The `Fl_Scheme_Choice` widget does this automatically when the widget is shown (when receiving the FL_SHOW event) which should always be after `Fl_Window::show(argc, argv)` which may set the current scheme by interpreting the commandline.

Since 1.4.0

### 12.121.2.3 scheme_cb()

```cpp
void Fl_Scheme_Choice::scheme_cb_ ( Fl_Widget * w, void * ) [static], [protected]
```

Internal `Fl_Scheme_Choice` callback function (protected). You don't need to set a callback for this widget. The default callback changes the scheme (`Fl::scheme()`) and redraws all open windows. You may override the callback if changing the scheme shall redraw other windows or don't redraw the window at all.

**Parameters**

| in | w | The Fl_Scheme_Choice widget |

The documentation for this class was generated from the following files:

- `Fl_Scheme_Choice.H`
- `Fl_Scheme_Choice.cxx`

### 12.122 Fl_Scroll Class Reference

This container widget lets you maneuver around a set of widgets much larger than your window.

```cpp
#include <Fl_Scroll.H>
```

Inheritance diagram for `Fl_Scroll`:

```
Fl_Widget
   /|
  / Fl_Group
 /|
Fl_Scroll
```
Classes

- **struct Fl_Region_LRTB**
  A local struct to manage a region defined by left/right/top/bottom.
- **struct Fl_Region_XYWH**
  A local struct to manage a region defined by xywh.
- **struct Fl_Scrollbar_Data**
  A local struct to manage a scrollbar’s xywh region and tab values.
- **struct ScrollInfo**
  Structure to manage scrollbar and widget interior sizes.

Public Types

- enum {
  HORIZONTAL = 1, VERTICAL = 2, BOTH = 3, ALWAYS_ON = 4,
  HORIZONTAL_ALWAYS = 5, VERTICAL_ALWAYS = 6, BOTH_ALWAYS = 7
}

Public Member Functions

- **void clear ()**
  Clear all but the scrollbars...
- **int delete_child (int n) FL_OVERRIDE**
  Removes the widget at index from the group and deletes it.
- **Fl_Scroll (int X, int Y, int W, int H, const char ∗L=0)**
  Creates a new Fl_Scroll widget using the given position, size, and label string.
- **int handle (int) FL_OVERRIDE**
  Handles the specified event.
- **void resize (int X, int Y, int W, int H) FL_OVERRIDE**
  Resizes the Fl_Scroll widget and moves its children if necessary.
- **void scroll_to (int, int)**
  Moves the contents of the scroll group to a new position.
- **int scrollbar_size () const**
  Gets the current size of the scrollbars’ troughs, in pixels.
- **void scrollbar_size (int newSize)**
  Sets the pixel size of the scrollbars’ troughs to newSize, in pixels.
- **int xposition () const**
  Gets the current horizontal scrolling position.
- **int yposition () const**
  Gets the current vertical scrolling position.
- **virtual ∼Fl_Scroll ()**
  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Group

- **Fl_Widget ∗& _ddfdesign_kludge ()**
  This is for forms compatibility only.
- **void add (Fl_Widget &)**
  The widget is removed from its current group (if any) and then added to the end of this group.
- **void add (Fl_Widget ∗o)**
  See void Fl_Group::add(Fl_Widget &w)
- **void add_resizable (Fl_Widget &o)**
  Adds a widget to the group and makes it the resizable widget.
- **Fl_Widget ∗const ∗ array () const**
Returns a pointer to the array of children.

- `Fl_Group const * as_group () const FL_OVERRIDE`  
  Returns an `Fl_Group` pointer if this widget is an `Fl_Group`.

- `void begin ()`  
  Sets the current group so you can build the widget tree by just constructing the widgets.

- `Fl_Widget * child (int n) const`  
  Returns `array()[n]`.

- `int children () const`  
  Returns how many child widgets the group has.

- `void clear ()`  
  Deletes all child widgets from memory recursively.

- `unsigned int clip_children ()`  
  Returns the current clipping mode.

- `void clip_children (int c)`  
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- `void end ()`  
  Exactly the same as `current(this->parent())`.

- `int find (const Fl_Widget &o) const`  
  See `int Fl_Group::find(const Fl_Widget *w) const`.

- `int find (const Fl_Widget *) const`  
  Searches the child array for the widget and returns the index.

- `Fl_Group (int, int, int, int, const char *)=0)`  
  Creates a new `Fl_Group` widget using the given position, size, and label string.

- `void focus (Fl_Widget *W)`

- `void forms_end ()`  
  This is for forms compatibility only.

- `void init_sizes ()`  
  Resets the internal array of widget sizes and positions.

- `void insert (Fl_Widget &, int i)`  
  The widget is removed from its current group (if any) and then inserted into this group.

- `void insert (Fl_Widget &o, Fl_Widget *before)`  
  This does `insert(w, find(before))`.

- `void remove (Fl_Widget &)`  
  Removes a widget from the group but does not delete it.

- `void remove (Fl_Widget *=o)`  
  Removes the widget `o` from the group.

- `void remove (int index)`  
  Removes the widget at index from the group but does not delete it.

- `Fl_Widget * resizable () const`  
  Returns the group's resizable widget.

- `void resizable (Fl_Widget &o)`  
  Sets the group's resizable widget.

- `void resizable (Fl_Widget *=o)`  
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- `virtual ~Fl_Group ()`  
  The destructor also deletes all the children.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ as_gl_window () const
- virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ as_window () const
- void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
  Checks if the widget value changed since the last callback.
- void clear_active ()
Marks the widget as inactive without sending events or changing focus.

- void **clear_changed** ()
  Marks the value of the widget as unchanged.

- void **clear_damage** (uchar c=0)
  Clears or sets the damage flags.

- void **clear_output** ()
  Sets a widget to accept input.

- void **clear_visible** ()
  Hides the widget.

- void **clear_visible_focus** ()
  Disables keyboard focus navigation with this widget.

- **Fl_Color** **color** () const
  Gets the background color of the widget.

- void **color** (**Fl_Color** bg)
  Sets the background color of the widget.

- void **color** (**Fl_Color** bg, **Fl_Color** sel)
  Sets the background and selection color of the widget.

- **Fl_Color** **color2** () const
  For back compatibility only.

- void **color2** (unsigned a)
  For back compatibility only.

- int **contains** (**const Fl_Widget** *w) const
  Checks if *w* is a child of this widget.

- void **copy_label** (**const char** *new_label)
  Sets the current label.

- void **copy_tooltip** (**const char** *text)
  Sets the current tooltip text.

- uchar **damage** () const
  Returns non-zero if **draw()** needs to be called.

- void **damage** (uchar c)
  Sets the damage bits for the widget.

- void **damage** (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- int **damage_resize** (int, int, int, int)
  Internal use only.

- void **deactivate** ()
  Deactivates the widget.

- **Fl_Image** * **deimage** ()
  Gets the image that is used as part of the widget label when in the inactive state.

- const **Fl_Image** * **deimage** () const
  Gets the image that is used as part of the widget label when in the inactive state.

- void **deimage** (**Fl_Image** &img)
  Sets the image to use as part of the widget label when in the inactive state.

- void **deimage** (**Fl_Image** *img)
  Sets the image to use as part of the widget label when in the inactive state.

- int **deimage_bound** () const
  Returns whether the inactive image is managed by the widget.

- void **do_callback** (**Fl_Callback_Reas**on reason=**FL_REASON_UNKNOWN**)
  Calls the widget callback function with default arguments.

- void **do_callback** (**Fl_Widget** *widget, long arg, **Fl_Callback_Reas**on reason=**FL_REASON_UNKNOWN**)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
Returns if a widget is used for output only.

- **Fl_Group** ∗ **parent ()** const
  
  Returns a pointer to the parent widget.

- **void** **parent (Fl_Group ∗p)**
  
  Internal use only - “for hacks only”.

- **void** **position (int X, int Y)**
  
  Repositions the window or widget.

- **void** **redraw ()**
  
  Schedules the drawing of the widget.

- **void** **redraw_label ()**
  
  Schedules the drawing of the label.

- **Fl_Color** **selection_color ()** const
  
  Gets the selection color.

- **void** **selection_color (Fl_Color a)**
  
  Sets the selection color.

- **void** **set_active ()**
  
  Marks the widget as active without sending events or changing focus.

- **void** **set_changed ()**
  
  Marks the value of the widget as changed.

- **void** **set_output ()**
  
  Sets a widget to output only.

- **void** **set_visible ()**
  
  Makes the widget visible.

- **void** **set_visible_focus ()**
  
  Enables keyboard focus navigation with this widget.

- **int** **shortcut_label ()** const
  
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.

- **void** **shortcut_label (int value)**
  
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.

- **virtual** **void** **show ()**
  
  Makes a widget visible.

- **void** **size (int W, int H)**
  
  Changes the size of the widget.

- **int** **take_focus ()**
  
  Gives the widget the keyboard focus.

- **unsigned int** **takesevents ()** const
  
  Returns if the widget is able to take events.

- **int** **testShortcut ()**
  
  Returns true if the widget’s label contains the entered ‘&x’ shortcut.

- **const char ∗** **tooltip ()** const
  
  Gets the current tooltip text.

- **void** **tooltip (const char ∗text)**
  
  Sets the current tooltip text.

- **Fl_Window ∗** **top_window ()** const
  
  Returns a pointer to the top-level window for the widget.

- **Fl_Window ∗** **top_window_offset (int &xoff, int &yoff)** const
  
  Finds the x/y offset of the current widget relative to the top-level window.

- **uchar** **type ()** const
  
  Gets the widget type.

- **void** **type (uchar t)**
  
  Sets the widget type.
• int \texttt{use\_accents\_menu} ()

  Returns non zero if MAC\_USE\_ACCENTS\_MENU flag is set, 0 otherwise.

• void \ast \texttt{user\_data} () const

  Gets the user data for this widget.

• void \texttt{user\_data} (FL Callback User Data \ast v, bool auto\_free)

  Sets the user data for this widget.

• void \texttt{user\_data} (void \ast v)

  Sets the user data for this widget.

• unsigned int \texttt{visible} () const

  Returns whether a widget is visible.

• unsigned int \texttt{visible\_focus} () const

  Checks whether this widget has a visible focus.

• void \texttt{visible\_focus} (int v)

  Modifies keyboard focus navigation.

• int \texttt{visible\_r} () const

  Returns whether a widget and all its parents are visible.

• int \texttt{w} () const

  Gets the widget width.

• FL\_When \texttt{when} () const

  Returns the conditions under which the callback is called.

• void \texttt{when} (uchar i)

  Sets the flags used to decide when a callback is called.

• FL\_Window \ast \texttt{window} () const

  Returns a pointer to the nearest parent window up the widget hierarchy.

• int \texttt{x} () const

  Gets the widget position in its window.

• int \texttt{y} () const

  Gets the widget position in its window.

• virtual \texttt{~FL\_Widget} ()

  Destroys the widget.

\textbf{Public Attributes}

• \texttt{FL\_Scrollbar hscrollbar}

• \texttt{FL\_Scrollbar scrollbar}

\textbf{Protected Member Functions}

• void \texttt{bbox} (int \&, int \&, int \&, int \&) const

  Returns the bounding box for the interior of the scrolling area, inside the scrollbars.

• void \texttt{draw} () \texttt{FL\_OVERRIDE}

  Draws the widget.

• void \texttt{fix\_scrollbar\_order} ()

  Ensure the scrollbars are the last children.

• int \texttt{on\_insert} (FL\_Widget \ast, int) \texttt{FL\_OVERRIDE}

  Change insert position of a child before it is added.

• int \texttt{on\_move} (int, int) \texttt{FL\_OVERRIDE}

  Change new position of a child before it is moved.

• void \texttt{recalc\_scrollbars} (ScrollInfo \&si) const

  Calculate visibility/size/position of scrollbars, find children's bounding box.
Protected Member Functions inherited from Fl_Group

- **Fl_Rect ∗ bounds ()**
  Returns the internal array of widget sizes and positions.

- **void draw_child (Fl_Widget &widget) const**
  Forces a child to redraw.

- **void draw_children ()**
  Draws all children of the group.

- **void draw_outside_label (const Fl_Widget &widget) const**
  Parents normally call this to draw outside labels of child widgets.

- **virtual void on_remove (int)**
  Allow derived groups to act when a child widget is removed from the group.

- **int ∗ sizes ()**
  Returns the internal array of widget sizes and positions.

- **void update_child (Fl_Widget &widget) const**
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- **void clear_flag (unsigned int c)**
  Clears a flag in the flags mask.

- **void draw_backdrop () const**
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- **void draw_box () const**
  Draws the widget box according its box style.

- **void draw_box (Fl_Boxtype t, Fl_Color c) const**
  Draws a box of type t, of color c at the widget's position and size.

- **void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const**
  Draws a box of type t, of color c at the position X,Y and size W,H.

- **void draw_focus () const**
  Draws a focus rectangle around the widget.

- **void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const**
  Draws a focus rectangle around the widget.

- **void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const**
  Draws a focus box for the widget at the given position and size.

- **void draw_label () const**
  Draws the widget's label at the defined label position.

- **void draw_label (int, int, int, int) const**
  Draws the label in an arbitrary bounding box.

- **Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)**
  Creates a widget at the given position and size.

- **unsigned int flags () const**
  Gets the widget flags mask.

- **void h (int v)**
  Internal use only.

- **void set_flag (unsigned int c)**
  Sets a flag in the flags mask.

- **void w (int v)**
  Internal use only.

- **void x (int v)**
  Internal use only.

- **void y (int v)**
  Internal use only.
Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()
  Returns the currently active group.
- static void current (Fl_Group *g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don’t set a callback.
- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
flags possible values enumeration.

12.122.1 Detailed Description

This container widget lets you maneuver around a set of widgets much larger than your window. If the child widgets are larger than the size of this object then scrollbars will appear so that you can scroll over to them:

![Figure 12.41 Fl_Scroll](image)

If all of the child widgets are packed together into a solid rectangle then you want to set box() to FL_NO_BOX or one of the _FRAME types. This will result in the best output. However, if the child widgets are a sparse arrangement you must set box() to a real _BOX type. This can result in some blinking during redrawing, but that can be solved by using a Fl_Double_Window.

The Fl_Scroll widget calculates the bounding box of all its children by using their widget positions and sizes (x, y, w, h). Outside labels are not considered. If you need outside labels of any widgets or free space outside of this bounding box you can add a tiny invisible Fl_Box at the relevant corner(s) of the Fl_Scroll widget, for instance:
By default you can scroll in both directions, and the scrollbars disappear if the data will fit in the area of the scroll. Use `Fl_Scroll::type()` to change this as follows:

- **0** - No scrollbars
- **Fl_Scroll::HORIZONTAL** - Only a horizontal scrollbar.
- **Fl_Scroll::VERTICAL** - Only a vertical scrollbar.
- **Fl_Scroll::BOTH** - The default is both scrollbars.
- **Fl_Scroll::HORIZONTAL_ALWAYS** - Horizontal scrollbar always on, vertical always off.
- **Fl_Scroll::VERTICAL_ALWAYS** - Vertical scrollbar always on, horizontal always off.
- **Fl_Scroll::BOTH_ALWAYS** - Both always on.

Use `scrollbar.align(int)` (see `void Fl_Widget::align(Fl_Align)`): to change what side the scrollbars are drawn on.

If the FL_ALIGN_LEFT bit is on, the vertical scrollbar is on the left. If the FL_ALIGN_TOP bit is on, the horizontal scrollbar is on the top. Note that only the alignment flags in scrollbar are considered. The flags in hscrollbar however are ignored.

This widget can also be used to pan around a single child widget "canvas". This child widget should be of your own class, with a `draw()` method that draws the contents. The scrolling is done by changing the x() and y() of the widget, so this child must use the x() and y() to position its drawing. To speed up drawing it should test `fl_not_clipped(int x, int y, int w, int h)` to find out if a particular area of the widget must be drawn.

Another very useful child is a single `Fl_Pack`, which is itself a group that packs its children together and changes size to surround them. Filling the `Fl_Pack` with `Fl_Tabs` groups (and then putting normal widgets inside those) gives you a very powerful scrolling list of individually-openable panels.

Fluid lets you create these, but you can only lay out objects that fit inside the `Fl_Scroll` without scrolling. Be sure to leave space for the scrollbars, as Fluid won’t show these either.

You cannot use `Fl_Window` as a child of this since the clipping is not conveyed to it when drawn, and it will draw over the scrollbars and neighboring objects.

### 12.122.2 Constructor & Destructor Documentation

#### 12.122.2.1 `Fl_Scroll()`

```cpp
Fl_Scroll::Fl_Scroll ( int X, int Y, int W, int H, const char ∗L = 0 )
```

Creates a new `Fl_Scroll` widget using the given position, size, and label string.

The default boxtype is FL_NO_BOX.

The destructor also deletes all the children. This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the `Fl_Scroll` and all of its children can be automatic (local) variables, but you must declare the `Fl_Scroll` first, so that it is destroyed last.

#### 12.122.2.2 `~Fl_Scroll()`

```cpp
Fl_Scroll::~Fl_Scroll ( ) [virtual]
```

The destructor also deletes all the children.

See also

```cpp
Fl_Group::~Fl_Group()
```
12.122.3 Member Function Documentation

12.122.3.1 bbox()

```c++
void Fl_Scroll::bbox (
    int & X,
    int & Y,
    int & W,
    int & H ) const [protected]
```

Returns the bounding box for the interior of the scrolling area, inside the scrollbars.
This method does not change the scrollbars or their visibility. First the scrollbar positions and visibility are calculated
as they should be, according to the positions and sizes of the children. Then the bounding box is calculated.
You may need to call `redraw()` to make sure the widget gets updated.

See also

recalc_scrollbars()

12.122.3.2 delete_child()

```c++
int Fl_Scroll::delete_child ( int index ) [virtual]
```

Removes the widget at `index` from the group and deletes it.
This method does nothing if `index` is out of bounds or if `Fl_Group::child(index)` is one of the scrollbars.

Parameters

<table>
<thead>
<tr>
<th>index</th>
</tr>
</thead>
<tbody>
<tr>
<td>index of child to be removed</td>
</tr>
</tbody>
</table>

Returns

success (0) or error code

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>success</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>index out of range</td>
</tr>
<tr>
<td>2</td>
<td>widget not allowed to be removed (see note)</td>
</tr>
</tbody>
</table>

See also

`Fl_Group::delete_child(int index)`

Since

FLTK 1.4.0

Reimplemented from `Fl_Group`.

12.122.3.3 draw()

```c++
void Fl_Scroll::draw ( ) [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as
soon as possible, call `redraw()` instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded
scrollbar, you can do it (because `draw()` is virtual) like this:

```c++
Fl_Widget *s = scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from `Fl_Group`. 

Generated by Doxygen
12.122.3.4 **fix_scrollbar_order()**

```c
void Fl_Scroll::fix_scrollbar_order ( ) [protected]
```

Ensure the scrollbars are the last children. When `Fl_Scroll` is instantiated the first child of the `Fl_Group` is the vertical scrollbar `scrollbar` and the second child is the horizontal scrollbar `hscrollbar`. These two widgets must always be the last two widgets and in this order to guarantee the correct drawing order and event delivery. Since FLTK 1.4.0 the new method `on_insert()` modifies the insert position of other children if it would be after the scrollbars.

12.122.3.5 **handle()**

```c
int Fl_Scroll::handle ( int event ) [virtual]
```

Handles the specified event. You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval. One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class. In rare cases you may want to return 1 from your `handle()` method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the `handle()` method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

**Parameters**

| in  | event | the kind of event received |

**Return values**

- 0 if the event was not used or understood
- 1 if the event was used and can be deleted

**See also**

`Fl_Event`

Reimplemented from `Fl_Group`.

12.122.3.6 **on_insert()**

```c
int Fl_Scroll::on_insert ( Fl_Widget * candidate, int index ) [protected], [virtual]
```

Change insert position of a child before it is added. Fix insert position if the new child is planned to be inserted after the scrollbars. We can assume that the scrollbars are always the last two children! `Fl_Group` calls this when a new widget is added. We return the new index if the new widget would be added after the scrollbars.

**Parameters**

| in  | candidate | the candidate will be added to the child array_ after this method returns. |
| in  | index     | add the child at this position in the array_ |

Generated by Doxygen
Returns

- index to position the child as planned
- a new index to force the child to a different position
-1 to keep the group from adding the candidate

Version

1.4.0

See also

Fl_Group::on_insert(Fl_Widget *candidate, int index)

Reimplemented from Fl_Group.

12.122.3.7 on_move()

```c
int Fl_Scroll::on_move (int old_index, int new_index) [protected], [virtual]
```

Change new position of a child before it is moved.
Fix new position if the new child is planned to be moved after the scrollbars. We can assume that the scrollbars are always the last two children!
Fl_Group calls this when a widget is moved within the list of children. We return a new index if the widget would be moved after the scrollbars.

Parameters

<table>
<thead>
<tr>
<th>old_index</th>
<th>the current index of the child that will be moved</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_index</td>
<td>the new index of the child</td>
</tr>
</tbody>
</table>

Returns

new index, possibly corrected to avoid last two scrollbar entries

Reimplemented from Fl_Group.

12.122.3.8 recalc_scrollbars()

```c
void Fl_Scroll::recalc_scrollbars (ScrollInfo & si) const [protected]
```

Calculate visibility/size/position of scrollbars, find children's bounding box.
The `si` parameter will be filled with data from the calculations. Derived classes can make use of this call to figure out the scrolling area eg. during resize() handling.
This method does not change the scrollbars or their visibility. It calculates the scrollbar positions and visibility as they should be, according to the positions and sizes of the children.
You may need to call `redraw()` to make sure the widget gets updated.

Parameters

| in,out | si | - ScrollInfo structure, filled with data |

Generated by Doxygen
12.122.3.9 resize()

void Fl_Scroll::resize {
    int X,
    int Y,
    int W,
    int H ) [virtual]
Resizes the Fl_Scroll widget and moves its children if necessary.
The Fl_Scroll widget first resizes itself, and then it moves all its children if (and only if) the Fl_Scroll widget has been
moved. The children are moved by the same amount as the Fl_Scroll widget has been moved, hence all children
keep their relative positions.

Note
Fl_Scroll::resize() does not call Fl_Group::resize(), and child widgets are not resized.
Since children of an Fl_Scroll are not resized, the resizable() widget is ignored (if it is set).
The scrollbars are moved to their proper positions, as given by Fl_Scroll::scrollbar.align(), and switched on or off as
necessary.

Note
Due to current (FLTK 1.3.x) implementation constraints some of this may effectively be postponed until the
Fl_Scroll is drawn the next time. This may change in a future release.

See also
Fl_Group::resizable()
Fl_Widget::resize(int,int,int,int)
Reimplemented from Fl_Group.

12.122.3.10 scroll_to()

void Fl_Scroll::scroll_to {
    int X,
    int Y )
Moves the contents of the scroll group to a new position.
This is like moving the scrollbars of the Fl_Scroll around. For instance:
Fl_Scroll scroll (10,10,200,200);
Fl_Box b1 ( 10, 10,50,50,"b1"); // relative (x,y) = (0,0)
Fl_Box b2 ( 60, 60,50,50,"b2"); // relative (x,y) = (50,50)
Fl_Box b3 (60,110,50,50,"b3"); // relative (x,y) = (50,100)
// populate scroll with more children ...
scroll.end();
scroll.scroll_to(50,100);
will move the logical origin of the internal scroll area to (-50,-100) relative to the origin of the Fl_Scroll (10,10), i.e.
Fl_Box b3 will be visible in the top left corner of the scroll area.

12.122.3.11 scrollbar_size() [1/2]

int Fl_Scroll::scrollbar_size ( ) const [inline]
Gets the current size of the scrollbars' troughs, in pixels.
If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.

Returns
Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also
Fl::scrollbar_size(int)
scrollbar_size() [2/2]

```cpp
void Fl_Scroll::scrollbar_size (  
    int newSize ) [inline]
```

Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.

Normally you should not need this method, and should use `Fl::scrollbar_size(int)` instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.

Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare.

Setting `newSize` to the special value of 0 causes the widget to track the global `Fl::scrollbar_size()`, which is the default.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>newSize</th>
<th>Sets the scrollbar size in pixels.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>If 0 (default), scrollbar size tracks the global <code>Fl::scrollbar_size()</code></td>
</tr>
</tbody>
</table>

**See also**

`Fl::scrollbar_size()`

12.122.3.13 xposition()

```cpp
int Fl_Scroll::xposition ( ) const [inline]
```

Gets the current horizontal scrolling position.

12.122.3.14 yposition()

```cpp
int Fl_Scroll::yposition ( ) const [inline]
```

Gets the current vertical scrolling position.

The documentation for this class was generated from the following files:

- Fl_Scroll.H
- Fl_Scroll.cxx

12.123 Fl_Scrollbar Class Reference

The `Fl_Scrollbar` widget displays a slider with arrow buttons at the ends of the scrollbar.

```cpp
#include <Fl_Scrollbar.H>
```

Inheritance diagram for `Fl_Scrollbar`:

```
Fl_Widget
   \--- Fl_Valuator
       \--- Fl_Slider
           \--- Fl_Scrollbar
```
Public Member Functions

- **Fl_Scrollbar** (int X, int Y, int W, int H, const char *L=0)
  
  Creates a new Fl_Scrollbar widget with given position, size, and label.

- **int handle**(int) FL_OVERRIDE
  
  Handles the specified event.

- **int linesize**(int i) const
  
  Get the size of step, in lines, that the arrow keys move.

- **void linesize**(int i)
  
  This number controls how big the steps are that the arrow keys do.

- **int value**(int) const
  
  Gets the integer value (position) of the slider in the scrollbar.

- **int value**(int p)
  
  Sets the value (position) of the slider in the scrollbar.

- **int value**(int pos, int windowSize, int first_line, int total_lines)
  
  Sets the position, size and range of the slider in the scrollbar.

- **~Fl_Scrollbar**()
  
  Destroys the Scrollbar.

Public Member Functions inherited from Fl_Slider

- **void bounds**(double a, double b)
  
  Sets the minimum (a) and maximum (b) values for the valuator widget.

- **Fl_Slider**(int X, int Y, int W, int H, const char *L=0)
  
  Creates a new Fl_Slider widget using the given position, size, and label string.

- **Fl_Slider**(uchar t, int X, int Y, int W, int H, const char *L)
  
  Creates a new Fl_Slider widget using the given type, position, size, and label string.

- **int handle**(int) FL_OVERRIDE
  
  Handles the specified event.

- **int scrollvalue**(int pos, int size, int first, int total)
  
  Sets the size and position of the sliding knob in the box.

- **Fl_Boxtype slider**(int) const
  
  Gets the slider box type.

- **void slider**(Fl_Boxtype c)
  
  Sets the slider box type.

- **float slider_size**(int) const
  
  Get the dimensions of the moving piece of slider.

- **void slider_size**(double v)
  
  Set the dimensions of the moving piece of slider.

Public Member Functions inherited from Fl_Valuator

- **void bounds**(double a, double b)
  
  Sets the minimum (a) and maximum (b) values for the valuator widget.

- **double clamp**(double)
  
  Clamps the passed value to the valuator range.

- **virtual int format**(char *)
  
  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.

- **double increment**(double, int)
  
  Adds n times the step value to the passed value.

- **double maximum**(int) const
  
  Gets the maximum value for the valuator.
• void maximum (double a)
  *Sets the maximum value for the valuator.*

• double minimum () const
  *Gets the minimum value for the valuator.*

• void minimum (double a)
  *Sets the minimum value for the valuator.*

• void precision (int digits)
  *Sets the step value to 1.0 / 10^digits.*

• void range (double a, double b)
  *Sets the minimum and maximum values for the valuator.*

• double round (double)
  *Round the passed value to the nearest step increment.*

• double step () const
  *Gets or sets the step value.*

• void step (double a, int b)
  *See double Fl_Valuator::step() const*

• void step (double s)
  *See double Fl_Valuator::step() const.*

• void step (int a)
  *See double Fl_Valuator::step() const*

• double value () const
  *Gets the floating point(double) value.*

• int value (double)
  *Sets the current value.*

### Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()

• void _set_fullscreen ()

• void activate ()
  *Activates the widget.*

• unsigned int active () const
  *Returns whether the widget is active.*

• int active_r () const
  *Returns whether the widget and all of its parents are active.*

• Fl_Align align () const
  *Gets the label alignment.*

• void align (Fl_Align alignment)
  *Sets the label alignment.*

• long argument () const
  *Gets the current user data (long) argument that is passed to the callback function.*

• void argument (long v)
  *Sets the current user data (long) argument that is passed to the callback function.*

• virtual class Fl_Gl_Window * as_gl_window ()
  *Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.*

• virtual class Fl_Gl_Window const * as_gl_window () const

• virtual Fl_Group * as_group ()
  *Returns an Fl_Group pointer if this widget is an Fl_Group.*

• virtual Fl_Group const * as_group () const

• virtual Fl_Window * as_window ()

• virtual Fl_Window const * as_window () const
Returns an `Fl_Window` pointer if this widget is an `Fl_Window`.

- virtual `Fl_Window` const * as_window () const

Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage (Fl_Image *img)

Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image (Fl_Image *img)

Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)

Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- `Fl_Boxtype` box () const

Gets the box type of the widget.

- void box (Fl_Boxtype new_box)

Sets the box type for the widget.

- `Fl_Callback_p` callback () const

Gets the current callback function for the widget.

- void callback (Fl_Callback *cb)

Sets the current callback function for the widget.

- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback *cb, void *p)

Sets the current callback function and data for the widget.

- void callback (Fl_Callback0 *cb)

Sets the current callback function for the widget.

- void callback (Fl_Callback1 *cb, long p=0)

Sets the current callback function for the widget.

- unsigned int changed () const

Checks if the widget value changed since the last callback.

- void clear_active ()

Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()

Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)

Clears or sets the damage flags.

- void clear_output ()

Sets a widget to accept input.

- void clear_visible ()

Hides the widget.

- void clear_visible_focus ()

Disables keyboard focus navigation with this widget.

- `Fl_Color` color () const

Gets the background color of the widget.

- void color (Fl_Color bg)

Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)

Sets the background and selection color of the widget.

- `Fl_Color` color2 () const

For back compatibility only.

- void color2 (unsigned a)

For back compatibility only.

- int contains (const Fl_Widget *w) const
Checks if w is a child of this widget.

- **void copy_label (const char ∗new_label)**
  Sets the current label.

- **void copy_tooltip (const char ∗text)**
  Sets the current tooltip text.

- **uchar damage () const**
  Returns non-zero if draw() needs to be called.

- **void damage (uchar c)**
  Sets the damage bits for the widget.

- **void damage (uchar c, int x, int y, int w, int h)**
  Sets the damage bits for an area inside the widget.

- **int damage_resize (int, int, int, int)**
  Internal use only.

- **void deactivate ()**
  Deactivates the widget.

- **FL_Image ∗deimage ()**
  Gets the image that is used as part of the widget label when in the inactive state.

- **const FL_Image ∗deimage () const**
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (FL_Image &img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (FL_Image ∗img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const**
  Returns whether the inactive image is managed by the widget.

- **void do_callback (FL_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with default arguments.

- **void do_callback (FL_Widget ∗widget, long arg, FL_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void do_callback (FL_Widget ∗widget, void ∗arg=0, FL_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void draw_label (int, int, int, int, Fl_Align) const**
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h () const**
  Gets the widget height.

- **virtual void hide ()**
  Makes a widget invisible.

- **FL_Image ∗image ()**
  Gets the image that is used as part of the widget label when in the active state.

- **const FL_Image ∗image () const**
  Gets the image that is used as part of the widget label when in the active state.

- **void image (FL_Image &img)**
  Sets the image to use as part of the widget label when in the active state.

- **void image (FL_Image ∗img)**
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const**
  Returns whether the image is managed by the widget.

- **int inside (const FL_Widget ∗wgt) const**
  Checks if this widget is a child of wgt.

- **int is_label_copied () const**
  Returns whether the current label was assigned with copy_label().
- `const char∗ label () const`
  
  Gets the current label text.

- `void label (const char∗ text)`
  
  Sets the current label pointer.

- `void label (FL_Labeltype a, const char ∗b)`
  
  Shortcut to set the label text and type in one call.

- `FL_Color labelcolor () const`
  
  Gets the label color.

- `void labelcolor (FL_Color c)`
  
  Sets the label color.

- `FL_Font labelfont () const`
  
  Gets the font to use.

- `void labelfont (FL_Font f)`
  
  Sets the font to use.

- `FL_Fontsize labelsize () const`
  
  Gets the font size in pixels.

- `void labelsize (FL_Fontsize pix)`
  
  Sets the font size in pixels.

- `FL_Labeltype labelfontype () const`
  
  Gets the label type.

- `void labelfontype (FL_Labeltype a)`
  
  Sets the label type.

- `void measure_label (int &ww, int &hh) const`
  
  Sets width ww and height hh accordingly with the label size.

- `bool needs_keyboard () const`
  
  Returns whether this widget needs a keyboard.

- `void needs_keyboard (bool needs)`
  
  Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  
  Returns if a widget is used for output only.

- `FL_Group ∗ parent () const`
  
  Returns a pointer to the parent widget.

- `void parent (FL_Group ∗p)`
  
  Internal use only - "for hacks only".

- `void position (int X, int Y)`
  
  Repositions the window or widget.

- `void redraw ()`
  
  Schedules the drawing of the widget.

- `void redraw_label ()`
  
  Schedules the drawing of the label.

- `virtual void resize (int x, int y, int w, int h)`
  
  Changes the size or position of the widget.

- `FL_Color selection_color () const`
  
  Gets the selection color.

- `void selection_color (FL_Color a)`
  
  Sets the selection color.

- `void set_active ()`
  
  Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  
  Marks the value of the widget as changed.

- `void set_output ()`
  
  Marks the widget as output.
Sets a widget to output only.

- void set_visible ()
  
  *Makes the widget visible.*

- void set_visible_focus ()
  
  *Enables keyboard focus navigation with this widget.*

- int shortcut_label () const
  
  *Returns whether the widget's label uses '§' to indicate shortcuts.*

- void shortcut_label (int value)
  
  *Sets whether the widget's label uses '§' to indicate shortcuts.*

- virtual void show ()
  
  *Makes a widget visible.*

- void size (int W, int H)
  
  *Changes the size of the widget.*

- int take_focus ()
  
  *Gives the widget the keyboard focus.*

- unsigned int takesevents () const
  
  *Returns if the widget is able to take events.*

- int test_shortcut ()
  
  *Returns true if the widget's label contains the entered '§x' shortcut.*

- const char * tooltip () const
  
  *Gets the current tooltip text.*

- void tooltip (const char *text)
  
  *Sets the current tooltip text.*

- Fl_Window * top_window () const
  
  *Returns a pointer to the top-level window for the widget.*

- Fl_Window * top_window_offset (int &xoff, int &yoff) const
  
  *Finds the x/y offset of the current widget relative to the top-level window.*

- uchar type () const
  
  *Gets the widget type.*

- void type (uchar t)
  
  *Sets the widget type.*

- int use_accents_menu ()
  
  *Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.*

- void * user_data () const
  
  *Gets the user data for this widget.*

- void user_data (Fl_Callback_User_Data *v, bool auto_free)
  
  *Sets the user data for this widget.*

- void user_data (void *v)
  
  *Sets the user data for this widget.*

- unsigned int visible () const
  
  *Returns whether a widget is visible.*

- unsigned int visible_focus () const
  
  *Checks whether this widget has a visible focus.*

- void visible_focus (int v)
  
  *Modifies keyboard focus navigation.*

- int visible_r () const
  
  *Returns whether a widget and all its parents are visible.*

- int w () const
  
  *Gets the widget width.*

- Fl_When when () const
  
  *Returns the conditions under which the callback is called.*
• void when (uchar i)
  
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  
  Gets the widget position in its window.
• int y () const
  
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  
  Destroys the widget.

Protected Member Functions

• void draw () FL_OVERRIDE
  
  Draws the widget.

Protected Member Functions inherited from Fl_Slider

• void draw () FL_OVERRIDE
  
  Draws the widget.
• void draw (int, int, int, int)
• int handle (int, int, int, int)

Protected Member Functions inherited from Fl_Valuator

• Fl_Valuator (int X, int Y, int W, int H, const char * L)
  
  Creates a new Fl_Valuator widget using the given position, size, and label string.
• void handle_drag (double newvalue)
  
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
• void handle_push ()
  
  Stores the current value in the previous value.
• void handle_release ()
  
  Called after an FL_WHEN_RELEASE event is received and before the callback.
• int horizontal () const
  
  Tells if the valuator is an FL_HORIZONTAL one.
• double previous_value () const
  
  Gets the previous floating point value before an event changed it.
• void set_value (double v)
  
  Sets the current floating point value.
• double softclamp (double)
  
  Clamps the value, but accepts v if the previous value is not already out of range.
• virtual void value_damage ()
  
  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  
  Clears a flag in the flags mask.
• void draw_backdrop () const
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const
  
  Draws the widget box according its box style.
• void draw_box (FL_Boxtyle t, FL_Color c) const
Draws a box of type \texttt{t}, of color \texttt{c} at the widget's position and size.

- \textbf{void} \texttt{draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const}
  
  Draws a box of type \texttt{t}, of color \texttt{c} at the position \texttt{X,Y} and size \texttt{W,H}.

- \textbf{void} \texttt{draw_focus () const}
  
  Draws a focus rectangle around the widget.

- \textbf{void} \texttt{draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const}
  
  Draws a focus rectangle around the widget.

- \textbf{void} \texttt{draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const}
  
  Draws a focus box for the widget at the given position and size.

- \textbf{void} \texttt{draw_label () const}
  
  Draws the widget's label at the defined label position.

- \textbf{void} \texttt{draw_label (int, int, int, int) const}
  
  Draws the label in an arbitrary bounding box.

- \textbf{Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)}
  
  Creates a widget at the given position and size.

- \textbf{unsigned int flags () const}
  
  Gets the widget flags mask.

- \textbf{void} \texttt{h (int v)}
  
  Internal use only.

- \textbf{void} \texttt{set_flag (unsigned int c)}
  
  Sets a flag in the flags mask.

- \textbf{void} \texttt{w (int v)}
  
  Internal use only.

- \textbf{void} \texttt{x (int v)}
  
  Internal use only.

- \textbf{void} \texttt{y (int v)}
  
  Internal use only.

### Additional Inherited Members

#### Static Public Member Functions inherited from \texttt{Fl_Widget}

- \textbf{static void} \texttt{default_callback (Fl_Widget ∗widget, void ∗data)}
  
  The default callback for all widgets that don't set a callback.

- \textbf{static unsigned int} \texttt{label_shortcut (const char ∗t)}
  
  Returns the Unicode value of the \texttt{&x} shortcut in a given text.

- \textbf{static int} \texttt{test_shortcut (const char ∗, const bool require_alt=false)}
  
  Returns true if the given text \texttt{t} contains the entered \texttt{&x} shortcut.

#### Protected Types inherited from \texttt{Fl_Widget}

- \textbf{enum} {\n  \texttt{INACTIVE = 1}<0, \texttt{INVISIBLE = 1}<1, \texttt{OUTPUT = 1}<2, \texttt{NOBORDER = 1}<3, \texttt{FORCE_POSITION = 1}<4, \texttt{NON_MODAL = 1}<5, \texttt{SHORTCUT_LABEL = 1}<6, \texttt{CHANGED = 1}<7, \texttt{OVERRIDE = 1}<8, \texttt{VISIBLE_FOCUS = 1}<9, \texttt{COPIED_LABEL = 1}<10, \texttt{CLIP_CHILDREN = 1}<11, \texttt{MENU_WINDOW = 1}<12, \texttt{TOOLTIP_WINDOW = 1}<13, \texttt{MODAL = 1}<14, \texttt{NO_OVERLAY = 1}<15, \texttt{GROUP_RELATIVE = 1}<16, \texttt{COPIED_TOOLTIP = 1}<17, \texttt{FULLSCREEN = 1}<18, \texttt{MAC_USE_ACCENTS_MENU = 1}<19, \texttt{NEEDS_KEYBOARD = 1}<20, \texttt{IMAGE_BOUND = 1}<21, \texttt{DEIMAGE_BOUND = 1}<22, \texttt{AUTO_DELETE_USER_DATA = 1}<23, \texttt{MAXIMIZED = 1}<24, \texttt{USERFLAG3 = 1}<29, \texttt{USERFLAG2 = 1}<30, \texttt{USERFLAG1 = 1}<31}\n  \}

  flags possible values enumeration.
12.123.1 Detailed Description

The Fl_Scrollbar widget displays a slider with arrow buttons at the ends of the scrollbar. Clicking on the arrows move up/left and down/right by linesize(). Scrollbars also accept FL_SHORTCUT events: the arrows move by linesize(), and vertical scrollbars take Page Up/Down (they move by the page size minus linesize()) and Home/End (they jump to the top or bottom). Scrollbars have step(1) preset (they always return integers). If desired you can set the step() to non-integer values. You will then have to use casts to get at the floating-point versions of value() from Fl_Slider.

![Fl_Scrollbar](image)

Figure 12.42 Fl_Scrollbar

12.123.2 Constructor & Destructor Documentation

12.123.2.1 Fl_Scrollbar()

```cpp
Fl_Scrollbar::Fl_Scrollbar ( 
    int X, 
    int Y, 
    int W, 
    int H, 
    const char ∗L = 0 )
```

Creates a new Fl_Scrollbar widget with given position, size, and label. You need to do type(FL_HORIZONTAL) if you want a horizontal scrollbar.

12.123.3 Member Function Documentation

12.123.3.1 draw()

```cpp
void Fl_Scrollbar::draw ( ) [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```cpp
FL_Widget ∗s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

12.123.3.2 handle()

```cpp
int Fl_Scrollbar::handle ( 
    int event ) [virtual]
```

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in event | the kind of event received |

Return values

| 0 | if the event was not used or understood |
| 1 | if the event was used and can be deleted |

See also

FL_Event

Reimplemented from Fl_Widget.

12.123.3.3 linesize()

void Fl_Scrollbar::linesize (int i) [inline]

This number controls how big the steps are that the arrow keys do. In addition page up/down move by the size last sent to value() minus one linesize(). The default is 16.

12.123.3.4 value() [1/3]

int Fl_Scrollbar::value ( ) const [inline]

Gets the integer value (position) of the slider in the scrollbar. You can get the floating point value with Fl_Slider::value().

See also

Fl_Scrollbar::value(int p)
Fl_Scrollbar::value(int pos, int size, int first, int total)

12.123.3.5 value() [2/3]

int Fl_Scrollbar::value (int p) [inline]

Sets the value (position) of the slider in the scrollbar.

See also

Fl_Scrollbar::value()
Fl_Scrollbar::value(int pos, int size, int first, int total)

12.123.3.6 value() [3/3]

int Fl_Scrollbar::value (int p, int windowSize, int first_line, int total_lines) [inline]

Sets the position, size and range of the slider in the scrollbar.
You should call this every time your window changes size, your data changes size, or your scroll position changes (even if in response to a callback from this scrollbar). All necessary calls to `redraw()` are done. Calls `Fl_Slider::scrollvalue(int pos, int size, int first, int total)`.

The documentation for this class was generated from the following files:

- Fl_Scrollbar.H
- Fl_Scrollbar.cxx

### 12.124 FI_Scroll::FI_Scrollbar_Data Struct Reference

A local struct to manage a scrollbar's xywh region and tab values.

```cpp
#include <Fl_Scroll.H>
```

#### Public Attributes

- int **first**
  scrollbar tab's "number of first line"
- int **h**
- int **pos**
  scrollbar tab's "position of first line displayed"
- int **size**
  scrollbar tab's "size of window in lines"
- int **total**
  scrollbar tab's "total number of lines"
- int **w**
- int **x**
- int **y**

### 12.124.1 Detailed Description

A local struct to manage a scrollbar's xywh region and tab values.

The documentation for this struct was generated from the following file:

- Fl_Scroll.H

### 12.125 FI_Secret_Input Class Reference

The **FI_Secret_Input** class is a subclass of **FI_Input** that displays its input as a string of placeholders.

```cpp
#include <FI_Secret_Input.H>
```

Inheritance diagram for **FI_Secret_Input**:
Public Member Functions

- **Fl_Secret_Input** (int X, int Y, int W, int H, const char *l=0)
  Creates a new Fl_Secret_Input widget using the given position, size, and label string.

- **int handle** (int) FL_OVERRIDE
  Handles the specified event.

Public Member Functions inherited from Fl_Input

- **Fl_Input** (int, int, int, int, const char * =0)
  Creates a new Fl_Input widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Input_

- **int append** (const char *t, int l=0, char keep_selection=0)
  Append text at the end.

- **bool can_redo** () const
  Check if there is a redo action available.

- **bool can_undo** () const
  Check if the last operation can be undone.

- **int copy** (int clipboard)
  Put the current selection into the clipboard.

- **int copy_cuts** ()
  Copies the yank buffer to the clipboard.

- **Fl_Color cursor_color** () const
  Gets the color of the cursor.

- **void cursor_color** (Fl_Color n)
  Sets the color of the cursor.

- **int cut** ()
  Deletes the current selection.

- **int cut** (int a, int b)
  Deletes all characters between index a and b.

- **int cut** (int n)
  Deletes the next n bytes rounded to characters before or after the cursor.

- **double dvalue** () const
  Returns the widget text interpreted as a floating point number.

- **Fl_Input_** (int, int, int, int, const char *=0)
  Creates a new Fl_Input_ widget.

- **unsigned int index** (int i) const
  Returns the character at index i.

- **int input_type** () const
  Gets the input field type.
• **void input_type** (int t)
  Sets the input field type.

• **int insert** (const char *t, int l=0)
  Inserts text at the cursor position.

• **int insert_position** () const
  Gets the position of the text cursor.

• **int insert_position** (int p)
  Sets the cursor position and mark.

• **int insert_position** (int p, int m)
  Sets the index for the cursor and mark.

• **int ivalue** () const
  Returns the widget text interpreted as a signed integer.

• **int mark** () const
  Gets the current selection mark.

• **int mark** (int m)
  Sets the current selection mark.

• **int maximum_size** () const
  Gets the maximum length of the input field in characters.

• **void maximum_size** (int m)
  Sets the maximum length of the input field in characters.

• **int position** () const

• **int position** (int p)

• **int position** (int p, int m)
  Gets the read-only state of the input field.

• **void readonly** (int b)
  Sets the read-only state of the input field.

• **int readonly** () const
  Gets the read-only state of the input field.

• **int redo** ()
  Redo previous undo operation.

• **int replace** (int b, int e, const char *text, int ilen=0)
  Deletes text from b to e and inserts the new string text.

• **void resize** (int, int, int, int) FL_OVERRIDE
  Changes the size of the widget.

• **int shortcut** () const
  Return the shortcut key associated with this widget.

• **void shortcut** (int s)
  Sets the shortcut key associated with this widget.

• **int size** () const
  Returns the number of bytes in value().

• **void size** (int W, int H)
  Sets the width and height of this widget.

• **int static_value** (const char *)
  Changes the widget text.

• **int static_value** (const char *, int)
  Changes the widget text.

• **int tab_nav** () const
  Gets whether the Tab key causes focus navigation in multiline input fields or not.

• **void tab_nav** (int val)
  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.

• **Fl_Color textcolor** () const
  Gets the color of the text in the input field.
• **void textcolor (Fl_Color n)**
  Sets the color of the text in the input field.

• **Fl_Font textfont () const**
  Gets the font of the text in the input field.

• **void textfont (Fl_Font s)**
  Sets the font of the text in the input field.

• **Fl_Fontsize textsize () const**
  Gets the size of the text in the input field.

• **void textsize (Fl_Fontsize s)**
  Sets the size of the text in the input field.

• **int undo ()**
  Undoes previous changes to the text buffer.

• **const char * value () const**
  Returns the text displayed in the widget.

• **int value (const char * s)**
  Changes the widget text.

• **int value (const char * s, int i)**
  Changes the widget text.

• **int value (double value)**
  Changes the widget text to a floating point number ("%g").

• **int value (int value)**
  Changes the widget text to a signed integer number.

• **int wrap () const**
  Gets the word wrapping state of the input field.

• **void wrap (int b)**
  Sets the word wrapping state of the input field.

• **~Fl_Input ()**
  Destroys the widget.

**Public Member Functions inherited from Fl_Widget**

• **void _clear_fullscreen ()**

• **void _set_fullscreen ()**

• **void activate ()**
  Activates the widget.

• **unsigned int active () const**
  Returns whether the widget is active.

• **int active_r () const**
  Returns whether the widget and all of its parents are active.

• **Fl_Align align () const**
  Gets the label alignment.

• **void align (Fl_Align alignment)**
  Sets the label alignment.

• **long argument () const**
  Gets the current user data (long) argument that is passed to the callback function.

• **void argument (long v)**
  Sets the current user data (long) argument that is passed to the callback function.

• **virtual class Fl_Gl_Window * as_gl_window ()**
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• **virtual class Fl_Gl_Window const * as_gl_window () const**

• **virtual Fl_Group * as_group ()**
Returns an Fl_Group pointer if this widget is an Fl_Group.

- virtual Fl_Group const *as_group() const
- virtual Fl_Window *as_window()

Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const *as_window() const
- void bind_deimage(Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage(int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image(Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image(int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- Fl_Boxtype box() const
  Gets the box type of the widget.
- void box(Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback() const
  Gets the current callback function for the widget.
- void callback(Fl_Callback *cb)
  Sets the current callback function for the widget.
- void callback(Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback(Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.
- void callback(Fl_Callback0 *cb)
- void callback(Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed() const
  Checks if the widget value changed since the last callback.
- void clear_active()
  Marks the widget as inactive without sending events or changing focus.
- void clear_changed()
  Marks the value of the widget as unchanged.
- void clear_damage(uchar c=0)
  Clears or sets the damage flags.
- void clear_output()
  Sets a widget to accept input.
- void clear_visible()
  Hides the widget.
- void clear_visible_focus()
  Disables keyboard focus navigation with this widget.
- Fl_Color color() const
  Gets the background color of the widget.
- void color(Fl_Color bg)
  Sets the background color of the widget.
- void color(Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
- Fl_Color color2() const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.
• void copy_label (const char *new_label)
  Sets the current label.
• void copy_tooltip (const char *text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image *image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image *image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
Checks if this widget is a child of `wgt`.

- `int is_label_copied () const`
  Returns whether the current label was assigned with `copy_label()`.

- `const char * label () const`
  Gets the current label text.

- `void label (const char *text)`
  Sets the current label pointer.

- `void label (Fl_Labeltype a, const char *b)`
  Shortcut to set the label text and type in one call.

- `Fl_Color labelcolor () const`
  Gets the label color.

- `void labelcolor (Fl_Color c)`
  Sets the label color.

- `Fl_Font labelfont () const`
  Gets the font to use.

- `void labelfont (Fl_Font f)`
  Sets the font to use.

- `Fl_Fontsize labelsize () const`
  Gets the font size in pixels.

- `void labelsize (Fl_Fontsize pix)`
  Sets the font size in pixels.

- `Fl_Labeltype labeltype () const`
  Gets the label type.

- `void labeltype (Fl_Labeltype a)`
  Sets the label type.

- `void measure_label (int &ww, int &hh) const`
  Sets width `ww` and height `hh` accordingly with the label size.

- `bool needs_keyboard () const`
  Returns whether this widget needs a keyboard.

- `void needs_keyboard (bool needs)`
  Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  Returns if a widget is used for output only.

- `Fl_Group * parent () const`
  Returns a pointer to the parent widget.

- `void parent (Fl_Group *p)`
  Internal use only - "for hacks only".

- `void position (int X, int Y)`
  Repositions the window or widget.

- `void redraw ()`
  Schedules the drawing of the widget.

- `void redraw_label ()`
  Schedules the drawing of the label.

- `Fl_Color selection_color () const`
  Gets the selection color.

- `void selection_color (Fl_Color a)`
  Sets the selection color.

- `void set_active ()`
  Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  Marks the value of the widget as changed.
• void `set_output()`
  Sets a widget to output only.
• void `set_visible()`
  Makes the widget visible.
• void `set_visible_focus()`
  Enables keyboard focus navigation with this widget.
• int `shortcut_label()` const
  Returns whether the widget's label uses `&` to indicate shortcuts.
• void `shortcut_label(int value)`
  Sets whether the widget's label uses `&` to indicate shortcuts.
• virtual void `show()`
  Makes a widget visible.
• void `size(int W, int H)`
  Changes the size of the widget.
• int `take_focus()`
  Gives the widget the keyboard focus.
• unsigned int `takesevents()` const
  Returns if the widget is able to take events.
• int `testShortcut()`
  Returns true if the widget's label contains the entered `&x` shortcut.
• const char * `tooltip()` const
  Gets the current tooltip text.
• void `tooltip(const char *text)`
  Sets the current tooltip text.
• Fl_Window * `top_window()` const
  Returns a pointer to the top-level window for the widget.
• Fl_Window * `top_window_offset(int &xoff, int &yoff)` const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar `type()` const
  Gets the widget type.
• void `type(uchar t)`
  Sets the widget type.
• int `use_accents_menu()`
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * `user_data()` const
  Gets the user data for this widget.
• void `user_data(Fl_Callback_User_Data *v, bool auto_free)`
  Sets the user data for this widget.
• void `user_data(void *v)`
  Sets the user data for this widget.
• unsigned int `visible()` const
  Returns whether a widget is visible.
• unsigned int `visible_focus()` const
  Checks whether this widget has a visible focus.
• void `visible_focus(int v)`
  Modifies keyboard focus navigation.
• int `visible_r()` const
  Returns whether a widget and all its parents are visible.
• int `w()` const
  Gets the widget width.
Returns the conditions under which the callback is called.

• void when (uchar i)
  Sets the flags used to decide when a callback is called.

• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  Gets the widget position in its window.

• int y () const
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget * widget, void * data)
  The default callback for all widgets that don't set a callback.

• static unsigned int label_shortcut (const char * t)
  Returns the Unicode value of the '&x' shortcut in a given text.

• static int test_shortcut (const char * t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Static Public Attributes inherited from Fl_Input

• static const char * copy_menu_text = "Copy"
  [this text may be customized at run-time]

• static const char * cut_menu_text = "Cut"
  [this text may be customized at run-time]

• static const char * paste_menu_text = "Paste"
  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

• enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19, NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31}

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Input

• void draw () FL_OVERRIDE
  Draws the widget.

• int handle_key ()
  Handles a keystroke.
- **int handle_rmb ()**
  Handle right mouse button down events.

**Protected Member Functions inherited from Fl_Input_**

- **int apply_undo ()**
  Apply the current undo/redo operation.
- **void drawtext (int, int, int, int)**
  Draws the text in the passed bounding box.
- **void drawtext (int, int, int, int, bool draw_active)**
  Draws the text in the passed bounding box.
- **void handle_mouse (int, int, int, int, int keepmark=0)**
  Handles mouse clicks and mouse moves.
- **int handletext (int, int, int, int)**
  Handles all kinds of text field related events.
- **int line_end (int i) const**
  Finds the end of a line.
- **int line_start (int i) const**
  Finds the start of a line.
- **int linesPerPage ()**
  **void maybe_do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  **int up_down_position (int, int, int keepmark=0)**
  Moves the cursor to the column given by up_down_pos.
- **int word_end (int i) const**
  Finds the end of a word.
- **int word_start (int i) const**
  Finds the start of a word.
- **int xscroll () const**
- **int yscroll () const**
- **void yscroll (int yOffset)**

**Protected Member Functions inherited from Fl_Widget**

- **void clear_flag (unsigned int c)**
  Clears a flag in the flags mask.
- **void draw_backdrop () const**
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- **void draw_box () const**
  Draws the widget box according its box style.
- **void draw_box (Fl_Boxtype t, Fl_Color c) const**
  Draws a box of type t, of color c at the widget's position and size.
- **void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const**
  Draws a box of type t, of color c at the position X,Y and size W,H.
- **void draw_focus () const**
  Draws a focus rectangle around the widget.
- **void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const**
  Draws a focus rectangle around the widget.
- **void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const**
  Draws a focus box for the widget at the given position and size.
- **void draw_label () const**
  Draws the widget's label at the defined label position.
- **void draw_label (int, int, int, int) const**
12.125.1 Detailed Description

The Fl_Secret_Input class is a subclass of Fl_Input that displays its input as a string of placeholders. Depending on the platform this placeholder is either the asterisk ('\*') or the Unicode bullet character (U+2022). This subclass is usually used to receive passwords and other "secret" information.

12.125.2 Constructor & Destructor Documentation

12.125.2.1 Fl_Secret_Input()

Fl_Secret_Input::Fl_Secret_Input (int X, int Y, int W, int H, const char *l = 0L)

Creates a new Fl_Secret_Input widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX.

Inherited destructor destroys the widget and any value associated with it.

12.125.3 Member Function Documentation

12.125.3.1 handle()

int Fl_Secret_Input::handle (int event) [virtual]

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class.

In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in event | the kind of event received |

Generated by Doxygen
Return values

<table>
<thead>
<tr>
<th>0</th>
<th>if the event was not used or understood</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>

See also

Fl_Event

Reimplemented from Fl_Input.

The documentation for this class was generated from the following files:

- Fl_Secret_Input.H
- Fl_Input.cxx

12.126 Fl_Select_Browser Class Reference

The class is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space.

```
#include <Fl_Select_Browser.H>
```

Inheritance diagram for Fl_Select_Browser:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Browser_
  ↓
Fl_Browser
  ↓
Fl_Select_Browser
```

Public Member Functions

- Fl_Select_Browser (int X, int Y, int W, int H, const char ∗L=0)
  Creates a new Fl_Select_Browser widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Browser

- void add (const char ∗newtext, void ∗d=0)
  Adds a new line to the end of the browser.
- void bottomline (int line)
  Scrolls the browser so the bottom item in the browser is showing the specified line.
- void clear ()
  Removes all the lines in the browser.
- char column_char () const
  Gets the current column separator character.
- void column_char (char c)
  Sets the column separator to c.
- const int ∗ column_widths () const
  Gets the current column width array.
- void column_widths (const int ∗arr)
Sets the current array to `arr`.

- `void * data (int line) const`
  Returns the user data() for specified line.

- `void data (int line, void *d)`
  Sets the user data for specified line to `d`.

- `void display (int line, int val=1)`
  For back compatibility.

- `int displayed (int line) const`
  Returns non-zero if line has been scrolled to a position where it is being displayed.

- `Fl_Browser (int X, int Y, int W, int H, const char *L=0)`
  The constructor makes an empty browser.

- `char format_char () const`
  Gets the current format code prefix character, which by default is '@'.

- `void format_char (char c)`
  Sets the current format code prefix character to `c`.

- `void hide () FL_OVERRIDE`
  Hides the entire Fl_Browser widget – opposite of show().

- `void hide (int line)`
  Makes line invisible, preventing selection by the user.

- `Fl_Image * icon (int line) const`
  Returns the icon currently defined for line.

- `void icon (int line, Fl_Image *icon)`
  Set the image icon for line to the value icon.

- `void insert (int line, const char *newtext, void *d=0)`
  Insert a new entry whose label is newtext above given line, optional data d.

- `void lineposition (int line, Fl_Line_Position pos)`
  Updates the browser so that line is shown at position pos.

- `int load (const char *filename)`
  Clear the browser and reads the file, adding each line from the file to the browser.

- `void make_visible (int line)`
  Make the item at the specified line visible().

- `void middleline (int line)`
  Scrolls the browser so the middle item in the browser is showing the specified line.

- `void move (int to, int from)`
  Line from is removed and reinserted at to.

- `void remove (int line)`
  Remove entry for given line number, making the browser one line shorter.

- `void remove_icon (int line)`
  Removes the icon for line.

- `void replace (int a, const char *b)`
  For back compatibility only.

- `int select (int line, int val=1)`
  Sets the selection state of the item at line to the value val.

- `int selected (int line) const`
  Returns 1 if specified line is selected, 0 if not.

- `void show () FL_OVERRIDE`
  Shows the entire Fl_Browser widget – opposite of hide().

- `void show (int line)`
  Makes line visible, and available for selection by user.

- `int size () const`
  Returns how many lines are in the browser.
• **void** *size*(int W, int H)
  
  Swaps two browser lines a and b.

• **void** *swap*(int a, int b)
  
  Swaps two browser lines a and b.

• **const char** ∗ *text*(int line) const
  
  Returns the label text for the specified line.

• **void** *text*(int line, const char ∗ newtext)
  
  Sets the text for the specified line to newtext.

• **FL_Fontsize** *textsize*() const
  
  Gets the default text size (in pixels) for the lines in the browser.

• **void** *textsize*(FL_Fontsize newSize)
  
  Sets the default text size (in pixels) for the lines in the browser to newSize.

• **int** *topline*(int line)
  
  Scrolls the browser so the top item in the browser is showing the specified line.

• **int** *value*(int line)
  
  Returns non-zero if the specified line is visible, 0 if hidden.

• **∼Fl_Browser**()
  
  The destructor deletes all list items and destroys the browser.

### Public Member Functions inherited from Fl_Browser_

• **int** *deselect*(int docallbacks=0)
  
  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.

• **void** *display*(void ∗ item)
  
  Displays the item, scrolling the list as necessary.

• **int** *handle*(int event) FL_OVERRIDE
  
  Handles the event within the normal widget bounding box.

• **uchar** *has_scrollbar*(uchar) const
  
  Returns the current scrollbar mode, see Fl_Browser_::has_scrollbar(uchar)

• **void** *has_scrollbar*(uchar mode)
  
  Sets whether the widget should have scrollbars or not (default Fl_Browser_::BOTH).

• **int** *hposition*(int)
  
  Gets the horizontal scroll position of the list as a pixel position pos.

• **void** *hposition*(int)
  
  Sets the horizontal scroll position of the list to pixel position pos.

• **int** *linespacing*(int pixels)
  
  Add some space between browser lines.

• **int** *position*(int pos)
  
  Repositions and/or resizes the browser.

• **void** *scrollbar_left*( )
  
  Moves the vertical scrollbar to the lefthand side of the list.
• void scrollbar_right ()

Moves the vertical scrollbar to the right hand side of the list.

• int scrollbar_size () const

Gets the current size of the scrollbars' troughs, in pixels.

• void scrollbar_size (int newSize)

Sets the pixel size of the scrollbars' troughs to newSize, in pixels.

• int scrollbar_width () const

Returns the global value Fl::scrollbar_size().

• void scrollbar_width (int width)

Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

• int select (void ∗item, int val=1, int docallbacks=0)

Sets the selection state of item to val, and returns 1 if the state changed or 0 if it did not.

• int select_only (void ∗item, int docallbacks=0)

Selects item and returns 1 if the state changed or 0 if it did not.

• void sort (int flags=0)

Sort the items in the browser based on flags.

• Fl_Color textcolor () const

Gets the default text color for the lines in the browser.

• void textcolor (Fl_Color col)

Sets the default text color for the lines in the browser to color col.

• Fl_Font textfont () const

Gets the default text font for the lines in the browser.

• void textfont (Fl_Font font)

Sets the default text font for the lines in the browser to font.

• Fl_Fontsize textsize () const

Gets the default text size (in pixels) for the lines in the browser.

• void textsize (Fl_Fontsize newSize)

Sets the default text size (in pixels) for the lines in the browser to size.

• int vposition () const

Gets the vertical scroll position of the list as a pixel position pos.

• void vposition (int pos)

Sets the vertical scroll position of the list to pixel position pos.

Public Member Functions inherited from Fl_Group

• Fl_Widget ∗ & _ddfdesign_kludge ()

This is for forms compatibility only.

• void add (Fl_Widget ∗&)

The widget is removed from its current group (if any) and then added to the end of this group.

• void add (Fl_Widget &)

See void Fl_Group::add(Fl_Widget &w)

• void add_resizable (Fl_Widget ∗&)

Adds a widget to the group and makes it the resizable widget.

• Fl_Widget ∗const ∗ array () const

Returns a pointer to the array of children.

• Fl_Group ∗ as_group () const FL_OVERRIDE

Returns an Fl_Group pointer if this widget is an Fl_Group.

• void begin ()

Sets the current group so you can build the widget tree by just constructing the widgets.

• Fl_Widget ∗ child (int n) const
Returns array()[n].

- **int children()** const
  
  Returns how many child widgets the group has.

- **void clear()**
  
  Deletes all child widgets from memory recursively.

- **unsigned int clip_children()**
  
  Returns the current clipping mode.

- **void clip_children(int c)**
  
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- **virtual int delete_child(int n)**
  
  Removes the widget at index from the group and deletes it.

- **void end()**
  
  Exactly the same as current(this->parent()).

- **int find(const Fl_Widget &w) const**
  
  See int Fl_Group::find(const Fl_Widget *w) const.

- **int find(const Fl_Widget *) const**
  
  Searches the child array for the widget and returns the index.

- **Fl_Group(int, int, int, int, const char *)=0**
  
  Creates a new Fl_Group widget using the given position, size, and label string.

- **void focus(Fl_Widget *)**

- **void forms_end()**
  
  This is for forms compatibility only.

- **int handle(int) FL_OVERRIDE**
  
  Handles the specified event.

- **void init_sizes()**
  
  Resets the internal array of widget sizes and positions.

- **void insert(Fl_Widget &, int i)**
  
  The widget is removed from its current group (if any) and then inserted into this group.

- **void insert(Fl_Widget &, Fl_Widget *) before**
  
  This does insert(w, find(before)).

- **void remove(Fl_Widget &)**
  
  Removes a widget from the group but does not delete it.

- **void remove(Fl_Widget *o)**
  
  Removes the widget o from the group.

- **void remove(int index)**
  
  Removes the widget at index from the group but does not delete it.

- **Fl_Widget *resizable()** const
  
  Returns the group's resizable widget.

- **void resizable(Fl_Widget &o)**
  
  Sets the group's resizable widget.

- **void resizable(Fl_Widget *) o**
  
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- **void resize(int, int, int, int) FL_OVERRIDE**
  
  Resizes the Fl_Group widget and all of its children.

- **virtual ~Fl_Group()**
  
  The destructor also deletes all the children.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ as_gl_window () const
- virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ as_window () const
- void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
  Checks if the widget value changed since the last callback.
- void clear_active ()
Marks the widget as inactive without sending events or changing focus.

• void clear_changed ()
  Marks the value of the widget as unchanged.

• void clear_damage (uchar c=0)
  Clears or sets the damage flags.

• void clear_output ()
  Sets a widget to accept input.

• void clear_visible ()
  Hides the widget.

• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

• Fl_Color color () const
  Gets the background color of the widget.

• void color (Fl_Color bg)
  Sets the background color of the widget.

• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

• Fl_Color color2 () const
  For back compatibility only.

• void color2 (unsigned a)
  For back compatibility only.

• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.

• void copy_label (const char *new_label)
  Sets the current label.

• void copy_tooltip (const char *text)
  Sets the current tooltip text.

• uchar damage () const
  Returns non-zero if draw() needs to be called.

• void damage (uchar c)
  Sets the damage bits for the widget.

• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

• int damage_resize (int, int, int, int)
  Internal use only.

• void deactivate ()
  Deactivates the widget.

• Fl_Image * deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

• const Fl_Image * deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
- `void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  
  Calls the widget callback function with arbitrary arguments.

- `void draw_label (int, int, int, int, Fl_Align) const`
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- `int h () const`
  
  Gets the widget height.

- `Fl_Image * image ()`
  
  Gets the image that is used as part of the widget label when in the active state.

- `const Fl_Image * image () const`
  
  Gets the image that is used as part of the widget label when in the active state.

- `void image (Fl_Image &img)`
  
  Sets the image to use as part of the widget label when in the active state.

- `void image (Fl_Image *img)`
  
  Sets the image to use as part of the widget label when in the active state.

- `int image_bound () const`
  
  Returns whether the image is managed by the widget.

- `int inside (const Fl_Widget *wgt) const`
  
  Checks if this widget is a child of `wgt`.

- `int is_label_copied () const`
  
  Returns whether the current label was assigned with `copy_label()`.

- `const char * label () const`
  
  Gets the current label text.

- `void label (const char *text)`
  
  Sets the current label pointer.

- `void label (Fl_Labeltype a, const char *b)`
  
  Shortcut to set the label text and type in one call.

- `Fl_Color labelcolor () const`
  
  Gets the label color.

- `void labelcolor (Fl_Color c)`
  
  Sets the label color.

- `Fl_Font labelfont () const`
  
  Gets the font to use.

- `void labelfont (Fl_Font f)`
  
  Sets the font to use.

- `Fl_Fontsize labelsize () const`
  
  Gets the font size in pixels.

- `void labelsize (Fl_Fontsize pix)`
  
  Sets the font size in pixels.

- `Fl_Labeltype labeltype () const`
  
  Gets the label type.

- `void labeltype (Fl_Labeltype a)`
  
  Sets the label type.

- `void measure_label (int &ww, int &hh) const`
  
  Sets width `ww` and height `hh` accordingly with the label size.

- `bool needs_keyboard () const`
  
  Returns whether this widget needs a keyboard.

- `void needs_keyboard (bool needs)`
  
  Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  
  Returns if a widget is used for output only.

- `Fl_Group * parent () const`
Returns a pointer to the parent widget.

- void parent (Fl_Group ∗p)
  
  Internal use only - "for hacks only".

- void position (int X, int Y)
  
  Repositions the window or widget.

- void redraw ()
  
  Schedules the drawing of the widget.

- void redraw_label ()
  
  Schedules the drawing of the label.

- Fl_Color selection_color () const
  
  Gets the selection color.

- void selection_color (Fl_Color a)
  
  Sets the selection color.

- void set_active ()
  
  Marks the widget as active without sending events or changing focus.

- void set_changed ()
  
  Marks the value of the widget as changed.

- void set_output ()
  
  Sets a widget to output only.

- void set_visible ()
  
  Makes the widget visible.

- void set_visible_focus ()
  
  Enables keyboard focus navigation with this widget.

- int shortcut_label () const
  
  Returns whether the widget's label uses ' & ' to indicate shortcuts.

- void shortcut_label (int value)
  
  Sets whether the widget's label uses ' & ' to indicate shortcuts.

- void size (int W, int H)
  
  Changes the size of the widget.

- int take_focus ()
  
  Gives the widget the keyboard focus.

- unsigned int takesevents () const
  
  Returns if the widget is able to take events.

- int test_shortcut ()
  
  Returns true if the widget's label contains the entered ' & x ' shortcut.

- const char ∗tooltip () const
  
  Gets the current tooltip text.

- void tooltip (const char ∗text)
  
  Sets the current tooltip text.

- Fl_Window ∗top_window () const
  
  Returns a pointer to the top-level window for the widget.

- Fl_Window ∗top_window_offset (int &xoff, int &yoff) const
  
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const
  
  Gets the widget type.

- void type (uchar t)
  
  Sets the widget type.

- int use_accents_menu ()
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void ∗user_data () const
  
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void ∗v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window ∗window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ∼Fl_Widget ()
  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Browser

• enum Fl_Line_Position { TOP, BOTTOM, MIDDLE }
  For internal use only?

Public Types inherited from Fl_Browser_

• enum {  
  HORIZONTAL = 1, VERTICAL = 2, BOTH = 3, ALWAYS_ON = 4,  
  HORIZONTAL_ALWAYS = 5, VERTICAL_ALWAYS = 6, BOTH_ALWAYS = 7 }
  Values for has_scrollbar().

Static Public Member Functions inherited from Fl_Group

• static Fl_Group ∗current ()
  Returns the currently active group.
• static void current (Fl_Group ∗g)
  Sets the current group.
Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)
  The default callback for all widgets that don't set a callback.
- static unsigned int labelShortcut (const char ∗t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int testShortcut (const char ∗t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Public Attributes inherited from Fl_Browser_

- Fl_Scrollbar hscrollbar
  Horizontal scrollbar.
- Fl_Scrollbar scrollbar
  Vertical scrollbar.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19, NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
flags possible values enumeration.

Protected Member Functions inherited from Fl_Browser

- FL_BLINE ∗_remove (int line)
  Removes the item at the specified line.
- FL_BLINE ∗find_line (int line) const
  Returns the item for specified line.
- int full_height () const FL_OVERRIDE
  The height of the entire list of all visible() items in pixels.
- int incr_height () const FL_OVERRIDE
  The default 'average' item height (including inter-item spacing) in pixels.
- void insert (int line, FL_BLINE ∗item)
  Insert specified item above line.
- void ∗item_at (int line) const FL_OVERRIDE
  Return the item at specified line.
- void item_draw (void ∗item, int X, int Y, int W, int H) const FL_OVERRIDE
  Draws item at the position specified by X Y WH.
- void ∗item_first () const FL_OVERRIDE
  Returns the very first item in the list.
- int item_height (void ∗item) const FL_OVERRIDE
  Returns height of item in pixels.
- void ∗item_last () const FL_OVERRIDE

Returns the very last item in the list.

• void ∗ item_next (void ∗item) const FL_OVERRIDE
  Returns the next item after item.

• void ∗ item_prev (void ∗item) const FL_OVERRIDE
  Returns the previous item before item.

• void item_select (void ∗item, int val) FL_OVERRIDE
  Change the selection state of item to the value val.

• int item_selected (void ∗item) const FL_OVERRIDE
  See if item is selected.

• void item_swap (void ∗a, void ∗b) FL_OVERRIDE
  Swap the items a and b.

• const char ∗ item_text (void ∗item) const FL_OVERRIDE
  Returns the label text for item.

• int item_width (void ∗item) const FL_OVERRIDE
  Returns width of item in pixels.

• int lineno (void ∗item) const
  Returns line number corresponding to item, or zero if not found.

• void swap (FL_BLINE ∗a, FL_BLINE ∗b)
  Swap the two items a and b.

Protected Member Functions inherited from Fl_Browser_

• void bbox (int &X, int &Y, int &W, int &H) const
  Returns the bounding box for the interior of the list’s display window, inside the scrollbars.

• void deleting (void ∗item)
  This method should be used when item is being deleted from the list.

• int displayed (void ∗item) const
  Returns non-zero if item has been scrolled to a position where it is being displayed.

• void draw () FL_OVERRIDE
  Draws the list within the normal widget bounding box.

• void ∗ find_item (int ypos)
  This method returns the item under mouse y position ypos.

• Fl_Browser_ (int X, int Y, int W, int H, const char ∗L=0)
  The constructor makes an empty browser.

• virtual int full_width () const
  This method may be provided by the subclass to indicate the full width of the item list, in pixels.

• void inserting (void ∗a, void ∗b)
  This method should be used when an item is in the process of being inserted into the list.

• virtual int item_quick_height (void ∗item) const
  This method may be provided by the subclass to return the height of the item, in pixels.

• int leftedge () const
  This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.

• void new_list ()
  This method should be called when the list data is completely replaced or cleared.

• void redraw_line (void ∗item)
  This method should be called when the contents of item has changed, but not its height.

• void redraw_lines ()
  This method will cause the entire list to be redrawn.

• void replacing (void ∗a, void ∗b)
  This method should be used when item a is being replaced by item b.

• void ∗ selection () const
Returns the item currently selected, or NULL if there is no selection.

- **void swapping (void ∗a, void ∗b)**
  This method should be used when two items a and b are being swapped.

- **void ∗top () const**
  Returns the item that appears at the top of the list.

### Protected Member Functions inherited from Fl_Group

- **Fl_Rect ∗bounds ()**
  Returns the internal array of widget sizes and positions.

- **void draw () FL_OVERRIDE**
  Draws the widget.

- **void draw_child (Fl_Widget &widget) const**
  Forces a child to redraw.

- **void draw_children ()**
  Draws all children of the group.

- **void draw_outside_label (const Fl_Widget &widget) const**
  Parents normally call this to draw outside labels of child widgets.

- **virtual int on_insert (Fl_Widget ∗, int)**
  Allow derived groups to act when a widget is added as a child.

- **virtual int on_move (int, int)**
  Allow derived groups to act when a widget is moved within the group.

- **virtual void on_remove (int)**
  Allow derived groups to act when a child widget is removed from the group.

- **int ∗sizes ()**
  Returns the internal array of widget sizes and positions.

- **void update_child (Fl_Widget &widget) const**
  Draws a child only if it needs it.

### Protected Member Functions inherited from Fl_Widget

- **void clear_flag (unsigned int c)**
  Clears a flag in the flags mask.

- **void draw_backdrop () const**
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- **void draw_box () const**
  Draws the widget box according its box style.

- **void draw_box (Fl_Boxtype t, Fl_Color c) const**
  Draws a box of type t, of color c at the widget's position and size.

- **void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const**
  Draws a box of type t, of color c at the position X,Y and size W,H.

- **void draw_focus () const**
  Draws a focus rectangle around the widget.

- **void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const**
  Draws a focus rectangle around the widget.

- **void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const**
  Draws a focus box for the widget at the given position and size.

- **void draw_label () const**
  Draws the widget's label at the defined label position.

- **void draw_label (int, int, int, int) const**
  Draws the label in an arbitrary bounding box.

- **Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)**
12.127 Fl_Shared_Image Class Reference

This class supports caching, loading, and drawing of image files.

#include <Fl_Shared_Image.H>

Inheritance diagram for Fl_Shared_Image:

```
   Fl_Image
     ▼
      Fl_Shared_Image
```

Generated by Doxygen
Public Member Functions

- **Fl_Shared_Image** ∗ as_shared_image () FL_OVERRIDE
  
  Returns whether an image is an Fl_Shared_Image or not.

- **void** color_average (Fl_Color c, float i) FL_OVERRIDE
  
  Averages the colors in the image with the provided FLTK color value.

- **Fl_Image** ∗ copy ()
  
  Increments the reference counter and returns a pointer to itself.

- **Fl_Image** ∗ copy () const

- **Fl_Image** ∗ copy (int W, int H) const FL_OVERRIDE
  
  Return a shared image of this image with the requested size.

- **void** desaturate () FL_OVERRIDE
  
  Convert the image to gray scale.

- **void** draw (int X, int Y)

- **void** draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE
  
  Draw this image to the current graphics context.

- **const Fl_Image** ∗ image () const
  
  Returns a pointer to the internal Fl_Image object.

- **const char** ∗ name ()
  
  Returns the filename of the shared image.

- **int** original ()
  
  Returns whether this is an original image.

- **int** refcount ()
  
  Returns the number of references of this shared image.

- **void** release () FL_OVERRIDE
  
  Releases and possibly destroys (if refcount <= 0) a shared image.

- **virtual void** reload ()
  
  Reloads the shared image from disk.

- **void** uncache () FL_OVERRIDE
  
  Remove the cached device specific image data.

Public Member Functions inherited from Fl_Image

- **Fl_Image** ∗ copy () const
  
  Creates a copy of the image in the same size.

- **int** count () const
  
  Returns the number of data values associated with the image.

- **int** d () const
  
  Returns the image depth.

- **const char** ∗ const ∗ data () const
  
  Returns a pointer to the current image data array.

- **int** data_h () const
  
  Returns the height of the image data.

- **int** data_w () const
  
  Returns the width of the image data.

- **void** draw (int X, int Y)
  
  Draws the image to the current drawing surface.

- **int** fail () const
  
  Returns a value that is not 0 if there is currently no image available.

- **Fl_Image** (int W, int H, int D)
  
  The constructor creates an empty image with the specified width, height, and depth.

- **int** h () const

Generated by Doxygen
Returns the current image drawing height in FLTK units.

- void **inactive ()**
  
  The **inactive()** method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

- virtual void **label (Fl_Menu_Item ∗m)**
  
  This method is an obsolete way to set the image attribute of a menu item.

- virtual void **label (Fl_Widget ∗w)**
  
  This method is an obsolete way to set the image attribute of a widget or menu item.

- int **ld () const**
  
  Returns the current line data size in bytes.

- virtual void **scale (int width, int height, int proportional=1, int can_expand=0)**
  
  Sets the drawing size of the image.

- int **w () const**
  
  Returns the current image drawing width in FLTK units.

- virtual **~Fl_Image ()**
  
  The destructor is a virtual method that frees all memory used by the image.

**Static Public Member Functions**

- static void **add_handler (Fl_Shared_Handler f)**
  
  Adds a shared image handler, which is basically a test function for adding new image formats.

- static Fl_Shared_Imagex **find (const char ∗name, int W=0, int H=0)**
  
  Finds a shared image from its name and size specifications.

- static Fl_Shared_Imagex **get (const char ∗name, int W=0, int H=0)**
  
  Find or load an image that can be shared by multiple widgets.

- static Fl_Shared_Imagex **get (Fl_RGB_Imagex ∗rgb, int own_it=1)**
  
  Builds a shared image from a pre-existing Fl_RGB_Imagex.

- static Fl_Shared_Imagex **∗∗images ()**
  
  Returns the Fl_Shared_Imagex array.

- static int **num_images ()**
  
  Number of shared images in their various cached sizes.

- static void **remove_handler (Fl_Shared_Handler f)**
  
  Removes a shared image handler.

**Static Public Member Functions inherited from Fl_Image**

- static Fl_Labeltype **define_FL_IMAGE_LABEL ()**

- static Fl_RGB_Scaling **RGB_scaling ()**
  
  Returns the currently used RGB image scaling method.

- static void **RGB_scaling (Fl_RGB_Scaling)**
  
  Sets the RGB image scaling method used for copy(int, int).

- static Fl_RGB_Scaling **scaling_algorithm ()**
  
  Gets what algorithm is used when resizing a source image to draw it.

- static void **scaling_algorithm (Fl_RGB_Scaling algorithm)**
  
  Sets what algorithm is used when resizing a source image to draw it.
Protected Member Functions

• void add()
  Adds a shared image to the image pool.

• Fl_Shared_Image * copy_(int W, int H) const
  Create a resized copy of the image and wrap it into the share image class.

• Fl_Shared_Image()
  Creates an empty shared image.

• Fl_Shared_Image(const char *n, Fl_Image *img=0)
  Creates a shared image from its filename and its corresponding Fl_Image* img.

• void update()
  Update the dimensions of the shared images.

• virtual ~Fl_Shared_Image()
  The destructor frees all memory and server resources that are used by the image.

Protected Member Functions inherited from Fl_Image

• void d(int D)
  Sets the current image depth.

• void data(const char *const *p, int c)
  Sets the current data pointer and count of pointers in the array.

• void draw_empty(int X, int Y)
  The protected method draw_empty() draws a box with an X in it.

• int draw_scaled(int X, int Y, int W, int H)
  Draw the image to the current drawing surface rescaled to a given width and height.

• void h(int H)
  Sets the height of the image data.

• void ld(int LD)
  Sets the current line data size in bytes.

• void w(int W)
  Sets the width of the image data.

Static Protected Member Functions

• static int compare(Fl_Shared_Image **i0, Fl_Shared_Image **i1)
  Compares two shared images.

Static Protected Member Functions inherited from Fl_Image

• static void labeltype(const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)

• static void measure(const Fl_Label *lo, int &lw, int &lh)

Protected Attributes

• int alloc_image_
• Fl_Image * image_
• const char * name_
• int original_
• int refcount_
Static Protected Attributes

- static int alloc_handlers_ = 0
- static int alloc_images_ = 0
- static Fl_Shared_Handler * handlers_ = 0
- static Fl_Shared_Image ** images_ = 0
- static int num_handlers_ = 0
- static int num_images_ = 0

Friends

- class Fl_Graphics_Driver
- class Fl_JPEG_Image
- class Fl_PNG_Image
- class Fl_SVG_Image

Additional Inherited Members

Static Public Attributes inherited from Fl_Image

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_MEMORY_ACCESS = -4
- static const int ERR_NO_IMAGE = -1
- static bool register_images_done = false

True after fl_register_images() was called, false before.

12.127.1 Detailed Description

This class supports caching, loading, and drawing of image files. Most applications will also want to link against the fltk_images library and call the fl_register_images() function to support standard image formats such as BMP, GIF, JPEG, PNG, and SVG (unless the library was built with the option removing SVG support).

Images can be requested (loaded) with Fl_Shared_Image::get(), find(), and some other methods. All images are cached in an internal list of shared images and should be released when they are no longer needed. A refcount is used to determine if a released image is to be destroyed with delete.

See also

fl_register_image()
Fl_Shared_Image::get()
Fl_Shared_Image::find()
Fl_Shared_Image::release()

12.127.2 Constructor & Destructor Documentation

12.127.2.1 Fl_Shared_Image() [1/2]

Fl_Shared_Image::Fl_Shared_Image ( ) [protected]
Creates an empty shared image.
The constructors create a new shared image record in the image cache.
The constructors are protected and cannot be used directly from a program. Use the get() method instead.


12.127.2.2  Fl_Shared_Image()  [2/2]

Fl_Shared_Image::Fl_Shared_Image (  
    const char ∗ n,  
    Fl_Image ∗ img = 0 )  [protected]

Creates a shared image from its filename and its corresponding Fl_Image* img.  
The constructors create a new shared image record in the image cache.  
The constructors are protected and cannot be used directly from a program. Use the get() method instead.  

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>filename or pool name of the image, must be unique among shared images</td>
</tr>
<tr>
<td>img</td>
<td>the image that is made available using the name</td>
</tr>
</tbody>
</table>

12.127.3  ~Fl_Shared_Image()

Fl_Shared_Image::~Fl_Shared_Image ( )  [protected], [virtual]

The destructor frees all memory and server resources that are used by the image.  
The destructor is protected and cannot be used directly from a program. Use the Fl_Shared_Image::release() method instead.  

12.127.3  Member Function Documentation

12.127.3.1  add()

void Fl_Shared_Image::add ( )  [protected]

Adds a shared image to the image pool.  
This protected method adds an image to the pool, an ordered list of shared images. The pool is searched for a matching image whenever one is requested, for instance with Fl_Shared_Image::get() or Fl_Shared_Image::find().  
This method does not increase or decrease reference counts!  

12.127.3.2  add_handler()

void Fl_Shared_Image::add_handler (  
    Fl_Shared_Handler f )  [static]

Adds a shared image handler, which is basically a test function for adding new image formats.  
This function will be called when an Fl_Shared_Image is to be loaded (for instance with Fl_Shared_Image::get()) and the image type is not known to FLTK.  
All registered image handlers will be called in the order of registration. You should always call fl_register_images() before adding your own handlers - unless you need to override a known image file type which should be rare.  
See also  
Fl_Shared_Handler for more information of the function you need to define.  

12.127.3.3  as_shared_image()

Fl_Shared_Image ∗ Fl_Shared_Image::as_shared_image ( )  [inline], [virtual]

Returns whether an image is an Fl_Shared_Image or not.  
This virtual method returns a pointer to an Fl_Shared_Image if this object is an instance of Fl_Shared_Image or NULL if not. This can be used to detect if a given Fl_Image object is a shared image, i.e. derived from Fl_Shared_Image.  
Since  
1.4.0  

Reimplemented from Fl_Image.
12.127.3.4 color_average()

```cpp
void Fl_Shared_Image::color_average ( 
    Fl_Color c, 
    float i ) [virtual]
```

Averages the colors in the image with the provided FLTK color value. This method changes the pixel data of this specific image.

**Note**

It does not change any of the resized copies of this image, nor does it necessarily apply the color changes if this image is resized later.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>c</th>
<th>blend with this color</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i</td>
<td>blend fraction</td>
</tr>
</tbody>
</table>

See also

`Fl_Image::color_average(Fl_Color c, float i)`

Reimplemented from `Fl_Image`.

12.127.3.5 compare()

```cpp
int Fl_Shared_Image::compare ( 
    Fl_Shared_Image ** i0, 
    Fl_Shared_Image ** i1 ) [static], [protected]
```

Compares two shared images. The order of comparison is:

1. Image name, usually the filename used to load it
2. Image width
3. Image height

Binary search in a sorted array works only if we search for the same parameters that were also used for sorting. No special cases are possible here. `Fl_Shared_Image::find()` requires a search for an element with a matching name and the original_ flags set. This is not implemented via binary search, but by a simple run of the array inside `Fl_Shared_Image::find()`.

**Parameters**

| in | i0,i1 | image pointer pointer for sorting |

**Returns**

Whether the images match or their relative sort order (see text).

**Return values**

| 0  | the images match |
| <0 | Image i0 is less than image i1 |
| >0 | Image i0 is greater than image i1 |
12.127.3.6 copy() [1/2]

Fl_Image * Fl_Shared_Image::copy ()
Increments the reference counter and returns a pointer to itself.
When the image is no longer used, call Fl_Shared_Image::release().
To get a copy of the image data, call this->image()->copy() instead.

Returns
pointer to an Fl_Shared_Image that can be safely cast

12.127.3.7 copy() [2/2]

Fl_Image * Fl_Shared_Image::copy ( 
    int W,
    int H ) const [virtual]
Return a shared image of this image with the requested size.
This is the same as calling Fl_Shared_Image::get(this->name(), W, H).
If a shared image of the desired size already exists in the shared image pool, the existing image is returned
and no copy is made. But the reference counter is incremented. When the image is no longer used, call
Fl_Shared_Image::release().
To get a copy of the image data, call this->image()->copy(W, H) instead.

Parameters

| in  | W/H  | size of requested image |

Returns
pointer to an Fl_Shared_Image that can be safely cast, or NULL if the image can't be found and can't be
created.
Reimplemented from Fl_Image.

12.127.3.8 copy_()

Fl_Shared_Image * Fl_Shared_Image::copy_ ( 
    int W,
    int H ) const [protected]
Create a resized copy of the image and wrap it into the share image class.
This function is usually followed by a call to returned_image->add() to add the image to the
pool, andthis->refcounter_++ to make sure that the original shared image keeps a reference to the copy. Don't
call this function if an image of the given size is already in the pool.

Parameters

| in  | W/H  | new image size |

Returns
a new shared image pointer that is not yet in the pool

12.127.3.9 desaturate()

void Fl_Shared_Image::desaturate ( ) [virtual]
Convert the image to gray scale.
This method changes the pixel data of this specific image.
Note

It does not change any of the resized copies of this image, nor does it necessarily apply the color changes if this image is resized later.

See also

Fl_Image::desaturate()

Reimplemented from Fl_Image.

### 12.127.3.10 draw()

```cpp
def draw (int X, int Y, int W, int H, int cx = 0, int cy = 0)
```

Draw this image to the current graphics context.

**Parameters**

- **in X, Y, W, H**: draw at this position and size
- **in cx, cy**: image origin

Reimplemented from Fl_Image.

### 12.127.3.11 find()

```cpp
Fl_Shared_Image * find (const char * name, int W = 0, int H = 0)
```

Finds a shared image from its name and size specifications. This uses a binary search in the image cache.

If the image `name` exists with the exact width `W` and height `H`, then it is returned.

If `W == 0` and the image `name` exists with another size, then the original image with that `name` is returned.

In either case the refcount of the returned image is increased. The found image should be released with Fl_Shared_Image::release() when no longer needed.

An image is marked original if it was directly loaded from a file or from memory as opposed to copied and resized images.

This comparison is used in Fl_Shared_Image::find() to find an image that matches the requested one or to find the position where a new image should be entered into the sorted list of shared images.

It is used in two steps by Fl_Shared_Image::add():

1. search with exact width and height
2. if not found, search again with width = 0 (and height = 0)

The first step will only return a match if the image exists with the same width and height. The second step will match if there is an image marked original with the same name, regardless of width and height.

### 12.127.3.12 get() [1/2]

```cpp
Fl_Shared_Image * get (const char * name, int W = 0, int H = 0)
```

Generated by Doxygen
Find or load an image that can be shared by multiple widgets.
If the image exists with the requested size, this image will be returned.
If the image exists, but only with another size, then a new copy with the requested size (width \(W\) and height \(H\)) will be created as a resized copy of the original image. The new image is added to the internal list of shared images.
If the image does not yet exist, then a new image of the proper dimension is created from the filename \(name\). The original image from filename \(name\) is always added to the list of shared images in its original size. If the requested size differs, then the resized copy with width \(W\) and height \(H\) is also added to the list of shared images.

**Note**
If the sizes differ, then two images are created as mentioned above. This is intentional so the original image is cached and preserved. If you request the same image with another size later, then the original image will be found, copied, resized, and returned.

Shared JPEG and PNG images can also be created from memory by using their named memory access constructor. You should release() the image when you’re done with it.

**Parameters**

<table>
<thead>
<tr>
<th>name</th>
<th>name of the image</th>
</tr>
</thead>
<tbody>
<tr>
<td>(W,H)</td>
<td>desired size</td>
</tr>
</tbody>
</table>

**Returns**
the image at the requested size, or NULL if the image could not be found or generated

**See also**

- Fl_Shared_Image::find(const char *name, int \(W\), int \(H\))
- Fl_Shared_Image::release()
- Fl_JPEG_Image::Fl_JPEG_Image(const char *name, const unsigned char *data)
- Fl_PNG_Image::Fl_PNG_Image (const char *name_png, const unsigned char *buffer, int maxsize)

**12.127.3.13 get()**

\[
\text{Fl\_Shared\_Image} * \text{Fl\_Shared\_Image}::\text{get} ( \\
\quad \text{Fl\_RGB\_Image} * \text{rgb}, \\
\quad \text{int own\_it} = 1) \] [static]

Builds a shared image from a pre-existing Fl_RGB_Image.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>(rgb)</th>
<th>an Fl_RGB_Image used to build a new shared image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{inown_it})</td>
<td>1 if the shared image should delete (rgb) when it is itself deleted, 0 otherwise</td>
<td></td>
</tr>
</tbody>
</table>

**Version**

1.3.4

**12.127.3.14 image()**

\[
\text{const Fl\_Image} * \text{Fl\_Shared\_Image}::\text{image} () \text{ const } [\text{inline}] 
\]

Returns a pointer to the internal Fl_IMAGE object.
The output is a pointer to the internal image (Fl_IMAGE or subclass) which can be used to inspect or copy the image.
Do not try to modify the image! You can copy the image though if you want or need to change any attributes, size etc. If all you need to do is to resize the image you should use Fl_Shared_Image::copy(int, int) instead.

Note
The internal image (pointer) is protected for good reasons, e.g. to prevent access to the image so it can't be modified by user code. DO NOT cast away the ‘const’ attribute to modify the image.

User code should rarely need this method. Use with caution.

Returns
const Fl_Image* image, the internal Fl_Image

Since
1.4.0

12.127.3.15 images()
Fl_Shared_Image ** Fl_Shared_Image::images ( ) [static]
Returns the Fl_Shared_Image* array.

Returns
a pointer to an array of shared image pointers, sorted by name and size

See also
Fl_Shared_Image::num_images()

12.127.3.16 num_images()
int Fl_Shared_Image::num_images ( ) [static]
Number of shared images in their various cached sizes.

Returns
number of entries in the array

See also
Fl_Shared_Image::images()

12.127.3.17 original()
int Fl_Shared_Image::original ( ) [inline]
Returns whether this is an original image.
Images loaded from a file or from memory are marked original as opposed to images created as a copy of another image with different size (width or height).

Note
This is useful for debugging (rarely used in user code).

Since
FLTK 1.4.0
12.127.3.18 refcount()

```cpp
int Fl_Shared_Image::refcount () [inline]
```

Returns the number of references of this shared image. When reference is below 1, the image is deleted.

12.127.3.19 release()

```cpp
void Fl_Shared_Image::release () [virtual]
```

Releases and possibly destroys (if refcount <= 0) a shared image. In the latter case, it will reorganize the shared image array so that no hole will occur. Reimplemented from Fl_Image.

12.127.3.20 uncache()

```cpp
void Fl_Shared_Image::uncache () [virtual]
```

Remove the cached device specific image data.

See also

Fl_Image::uncache()

Reimplemented from Fl_Image.

12.127.3.21 update()

```cpp
void Fl_Shared_Image::update () [protected]
```

Update the dimensions of the shared images. Internal method to synchronize shared image data with the actual image data.

The documentation for this class was generated from the following files:

- Fl_Shared_Image.H
- Fl_Shared_Image.cxx

12.128 Fl_Shortcut_Button Class Reference

A button that allows the user to type a key combination to create shortcuts.

Inheritance diagram for Fl_Shortcut_Button:

```
Fl_Widget
    ^
  |   
Fl_Button
    ^
  |   
Fl_Shortcut_Button
```

Public Member Functions

- **Fl_Shortcut_Button** (int X, int Y, int W, int H, const char *l=0)

  *Construct a shortcut button.*

- **Fl_Shortcut value ()**

  *Return the user selected shortcut.*

- **void value (Fl_Shortcut shortcut)**

  *Set the displayed shortcut.*
Public Member Functions inherited from **Fl_Button**

- `int clear ()`
  
  *Same as `value(0)`.*

- `uchar compact ()`
  
  *Return true if buttons are rendered as compact buttons.*

- `void compact (uchar v)`
  
  *Decide if buttons should be rendered in compact mode.*

- `Fl_Boxtype down_box () const`
  
  *Returns the current down box type, which is drawn when `value() is non-zero.*

- `void down_box (Fl_Boxtype b)`
  
  *Sets the down box type.*

- `Fl_Color down_color () const`
  
  *(for backwards compatibility)*

- `void down_color (unsigned c)`
  
  *(for backwards compatibility)*

- `Fl_Button (int X, int Y, int W, int H, const char ∗L=0)`
  
  *The constructor creates the button using the given position, size, and label.*

- `int set ()`
  
  *Same as `value(1)`.*

- `void setonly ()`
  
  *Turns on this button and turns off all other radio buttons in the group (calling `value(1)` or `set()` does not do this).*

- `int shortcut () const`
  
  *Returns the current shortcut key for the button.*

- `void shortcut (const char ∗s)`
  
  *(for backwards compatibility)*

- `void shortcut (int s)`
  
  *Sets the shortcut key to s.*

- `char value () const`
  
  *Returns the current value of the button (0 or 1).*

- `int value (int v)`
  
  *Sets the current value of the button.*

Public Member Functions inherited from **Fl_Widget**

- `void _clear_fullscreen ()`

- `void _set_fullscreen ()`

- `void activate ()`
  
  *Activates the widget.*

- `unsigned int active () const`
  
  *Returns whether the widget is active.*

- `int active_r () const`
  
  *Returns whether the widget and all of its parents are active.*

- `Fl_Align align () const`
  
  *Gets the label alignment.*

- `void align (Fl_Align alignment)`
  
  *Sets the label alignment.*

- `long argument () const`
  
  *Gets the current user data (long) argument that is passed to the callback function.*

- `void argument (long v)`
  
  *Sets the current user data (long) argument that is passed to the callback function.*

- `virtual class Fl_Gl_Window ∗ as_gl_window ()`
Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const * as_gl_window () const

Returns an Fl_Group pointer if this widget is an Fl_Group.
• virtual Fl_Group const * as_group () const
• virtual Fl_Window * as_window ()

Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage (Fl_Image * img)

Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_deimage (int f)

Sets the image to use as part of the widget label when in the active state.
• void bind_image (Fl_Image * img)

Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image (int f)

Gets the box type of the widget.
• Fl_Boxtype box () const

Sets the box type for the widget.
• void box (Fl_Boxtype new_box)

Gets the current callback function for the widget.
• Fl_Callback_p callback () const

Sets the current callback function for the widget.
• void callback (Fl_Callback * cb)

Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback * cb, Fl_Callback_User_Data * p, bool auto_free)

Sets the current callback function and data for the widget.
• void callback (Fl_Callback * cb, void * p)

Sets the current callback function for the widget.
• void callback (Fl_Callback0 * cb)

Sets the current callback function for the widget.
• void callback (Fl_Callback1 * cb, long p=0)

Checks if the widget value changed since the last callback.
• unsigned int changed () const

Marks the widget as inactive without sending events or changing focus.
• void clear_active ()

Marks the value of the widget as unchanged.
• void clear_changed ()

Clears or sets the damage flags.
• void clear_damage (uchar c=0)

Sets a widget to accept input.
• void clear_output ()

Hides the widget.
• void clear_visible ()

Disables keyboard focus navigation with this widget.
• void clear_visible_focus ()

Gets the background color of the widget.
• Fl_Color color () const

Sets the background color of the widget.
• void color (Fl_Color bg)

Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
Sets the background and selection color of the widget.

- **FL_Color color2 () const**
  For back compatibility only.

- **void color2 (unsigned a)**
  For back compatibility only.

- **int contains (const FL_Widget *w) const**
  Checks if w is a child of this widget.

- **void copy_label (const char *new_label)**
  Sets the current label.

- **void copy_tooltip (const char *text)**
  Sets the current tooltip text.

- **uchar damage () const**
  Returns non-zero if draw() needs to be called.

- **void damage (uchar c)**
  Sets the damage bits for the widget.

- **void damage (uchar c, int x, int y, int w, int h)**
  Sets the damage bits for an area inside the widget.

- **int damage_resize (int, int, int, int)**
  Internal use only.

- **void deactivate ()**
  Deactivates the widget.

- **FL_Image * deimage ()**
  Gets the image that is used as part of the widget label when in the inactive state.

- **const FL_Image * deimage () const**
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (FL_Image &img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (FL_Image *img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const**
  Returns whether the inactive image is managed by the widget.

- **void do_callback (FL_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with default arguments.

- **void do_callback (FL_Widget *widget, long arg, FL_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void do_callback (FL_Widget *widget, void *arg=0, FL_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with arbitrary arguments.

- **void draw_label (int, int, int, int, FL_Align) const**
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h () const**
  Gets the widget height.

- **virtual void hide ()**
  Makes a widget invisible.

- **FL_Image * image ()**
  Gets the image that is used as part of the widget label when in the active state.

- **const FL_Image * image () const**
  Gets the image that is used as part of the widget label when in the active state.

- **void image (FL_Image &img)**
  Sets the image to use as part of the widget label when in the active state.

- **void image (FL_Image *img)**
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget ∗wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char ∗ label () const
  Gets the current label text.
• void label (const char ∗text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char ∗b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group ∗ parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group ∗p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
Sets the selection color.

- void set_active ()
  Marks the widget as active without sending events or changing focus.

- void set_changed ()
  Marks the value of the widget as changed.

- void set_output ()
  Sets a widget to output only.

- void set_visible ()
  Makes the widget visible.

- void set_visible_focus ()
  Enables keyboard focus navigation with this widget.

- int shortcut_label () const
  Returns whether the widget's label uses ' & ' to indicate shortcuts.

- void shortcut_label (int value)
  Sets whether the widget's label uses ' & ' to indicate shortcuts.

- virtual void show ()
  Makes a widget visible.

- void size (int W, int H)
  Changes the size of the widget.

- int take_focus ()
  Gives the widget the keyboard focus.

- unsigned int takesevents () const
  Returns if the widget is able to take events.

- int test_shortcut ()
  Returns true if the widget's label contains the entered ' & x ' shortcut.

- const char ∗ tooltip () const
  Gets the current tooltip text.

- void tooltip (const char ∗ text)
  Sets the current tooltip text.

- Fl_Window ∗ top_window () const
  Returns a pointer to the top-level window for the widget.

- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const
  Gets the widget type.

- void type (uchar t)
  Sets the widget type.

- int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void ∗ user_data () const
  Gets the user data for this widget.

- void user_data (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.

- void user_data (void ∗v)
  Sets the user data for this widget.

- unsigned int visible () const
  Returns whether a widget is visible.

- unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

- void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.

• int w () const
  Gets the widget width.

• Fl_When when () const
  Returns the conditions under which the callback is called.

• void when (uchar i)
  Sets the flags used to decide when a callback is called.

• Fl_Window ∗ window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  Gets the widget position in its window.

• int y () const
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  Destroys the widget.

Protected Member Functions

• void do_end_hot_callback ()
  Call the callback if the user is interested.

• void draw () FL_OVERRIDE
  Draw the textual representation of the shortcut button.

• int handle (int) FL_OVERRIDE
  Handle keystrokes to catch the user's shortcut.

Protected Member Functions inherited from Fl_Button

• void simulate_key_action ()

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.

• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• void draw_box () const
  Draws the widget box according its box style.

• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

• void draw_focus () const
  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

• void draw_label () const
  Draws the widget's label at the defined label position.

• void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
• `Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)`
  
  Creates a widget at the given position and size.

• unsigned int flags () const

  Gets the widget flags mask.

• void h (int v)

  Internal use only.

• void set_flag (unsigned int c)

  Sets a flag in the flags mask.

• void w (int v)

  Internal use only.

• void x (int v)

  Internal use only.

• void y (int v)

  Internal use only.

Protected Attributes

• `Fl_Shortcut shortcut_value`

Additional Inherited Members

Static Public Member Functions inherited from `Fl_Widget`

• static void default_callback (Fl_Widget ∗widget, void ∗data)

  The default callback for all widgets that don’t set a callback.

• static unsigned int label_shortcut (const char ∗t)

  Returns the Unicode value of the '&x' shortcut in a given text.

• static int test_shortcut (const char ∗t, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from `Fl_Widget`

• enum {

  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Static Protected Member Functions inherited from `Fl_Button`

• static void key_release_timeout (void ∗)

Static Protected Attributes inherited from `Fl_Button`

• static Fl_Widget_Tracker ∗key_release_tracker = 0

Generated by Doxygen
12.128.1 Detailed Description

A button that allows the user to type a key combination to create shortcuts. After clicked once, the button catches the following keyboard events and records the pressed keys and all modifiers. It draws a text representation of the shortcut. The backspace key deletes the current shortcut. A second click on the button or moving focus makes the last shortcut permanent.

The Shortcut button calls the user callback after every change if FL_WHEN_CHANGED is set, and when the button is no longer recording shortcuts if FL_WHEN_RELEASE is set.

12.128.2 Constructor & Destructor Documentation

12.128.2.1 Fl_Shortcut_Button()

```cpp
Fl_Shortcut_Button::Fl_Shortcut_Button ( 
    int X, 
    int Y, 
    int W, 
    int H, 
    const char * l = 0 )
```

Construct a shortcut button.

**Parameters**

<table>
<thead>
<tr>
<th>X, Y, W, H</th>
<th>position and size of the button</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>label text when no shortcut is set</td>
</tr>
</tbody>
</table>

12.128.3 Member Function Documentation

12.128.3.1 draw()

```cpp
void Fl_Shortcut_Button::draw ( 
    void ) [protected], [virtual]
```

Draw the textual representation of the shortcut button. When the button can receive shortcut key events, it's "hot". A hot button is drawn in selection color. A cold button is drawn as a regular text box containing a human readable version of the shortcut key. Reimplemented from Fl_Button.

12.128.3.2 handle()

```cpp
int Fl_Shortcut_Button::handle ( 
    int e ) [protected], [virtual]
```

Handle keystrokes to catch the user's shortcut. Reimplemented from Fl_Button.

12.128.3.3 value() [1/2]

```cpp
Fl_Shortcut Fl_Shortcut_Button::value ( )
```

Return the user selected shortcut. Returns

**shortcut encoded as key and modifier**

12.128.3.4 value() [2/2]

```cpp
void Fl_Shortcut_Button::value ( 
    Fl_Shortcut shortcut )
```

Set the displayed shortcut.
Parameters

| in | shortcut | encoded as key and modifier |

The documentation for this class was generated from the following files:

- Fl_Shortcut_Button.H
- Fl_Shortcut_Button.cxx

### 12.129 Fl_Simple_Counter Class Reference

This widget creates a counter with only 2 arrow buttons.

```cpp
#include <Fl_Simple_Counter.H>
```

Inheritance diagram for Fl_Simple_Counter:

```
Fl_Widget
   
Fl_Valuator
   
Fl_Counter
   
Fl_Simple_Counter
```

#### Public Member Functions

- `Fl_Simple_Counter(int X, int Y, int W, int H, const char *L=0)`

#### Public Member Functions inherited from Fl_Counter

- `Fl_Counter(int X, int Y, int W, int H, const char *L=0)`
  - Creates a new Fl_Counter widget using the given position, size, and label string.
- `int handle(int) FL_OVERRIDE`
  - Handles the specified event.
- `void lstep(double a)`
  - Sets the increment for the large step buttons.
- `double step() const`
  - Returns the increment for normal step buttons.
- `void step(double a)`
  - Sets the increment for the normal step buttons.
- `void step(double a, double b)`
  - Sets the increments for the normal and large step buttons.
- `Fl_Color textcolor() const`
  - Gets the font color.
- `void textcolor(Fl_Color s)`
  - Sets the font color to s.
- `Fl_Font textfont() const`
  - Gets the text font.
- `void textfont(Fl_Font s)`
  - Sets the text font to s.
- `Fl_Fontsize textsize() const`
  - Gets the font size.
• void `textsize` (Fl_Fontsize s)
  Sets the font size to s.
• `~Fl_Counter` ()
  Destroys the valuator.

Public Member Functions inherited from `Fl_Valuator`

• void `bounds` (double a, double b)
  Sets the minimum (a) and maximum (b) values for the valuator widget.
• double `clamp` (double)
  Clamps the passed value to the valuator range.
• virtual int `format` (char *)
  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
• double `increment` (double, int)
  Adds n times the step value to the passed value.
• double `maximum` () const
  Gets the maximum value for the valuator.
• void `maximum` (double a)
  Sets the maximum value for the valuator.
• double `minimum` () const
  Gets the minimum value for the valuator.
• void `minimum` (double a)
  Sets the minimum value for the valuator.
• void `precision` (int digits)
  Sets the step value to 1.0 / 10^digits.
• void `range` (double a, double b)
  Sets the minimum and maximum values for the valuator.
• double `round` (double)
  Round the passed value to the nearest step increment.
• double `step` () const
  Gets or sets the step value.
• void `step` (double a, int b)
  See double `Fl_Valuator::step()` const
• void `step` (double s)
  See double `Fl_Valuator::step()` const.
• void `step` (int a)
  See double `Fl_Valuator::step()` const
• double `value` () const
  Gets the floating point(double) value.
• int `value` (double)
  Sets the current value.

Public Member Functions inherited from `Fl_Widget`

• void `_clear_fullscreen` ()
• void `_set_fullscreen` ()
• void `activate` ()
  Activates the widget.
• unsigned int `active` () const
  Returns whether the widget is active.
• int active_r () const
  Returns whether the widget and all of its parents are active.

• Fl_Align align () const
  Gets the label alignment.

• void align (Fl_Align alignment)
  Sets the label alignment.

• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.

• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const * as_gl_window () const
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual Fl_Group * as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Group const * as_group () const
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window const * as_window () const
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• void bind_deimage (Fl_Image * img)
  Sets the image to use as part of the widget label when in the inactive state.

• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

• void bind_image (Fl_Image * img)
  Sets the image to use as part of the widget label when in the active state.

• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

• Fl_Boxtype box () const
  Gets the box type of the widget.

• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.

• Fl_Callback_p callback () const
  Gets the current callback function for the widget.

• void callback (Fl_Callback * cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback * cb, Fl_Callback_User_Data * p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

• void callback (Fl_Callback * cb, void * p)
  Sets the current callback function and data for the widget.

• void callback (Fl_Callback0 * cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback1 * cb, long p=0)
  Sets the current callback function for the widget.

• unsigned int changed () const
  Checks if the widget value changed since the last callback.

• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

• void clear_changed ()
  Marks the value of the widget as unchanged.

• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget ∗ w) const
  Checks if w is a child of this widget.
• void copy_label (const char ∗ new_label)
  Sets the current label.
• void copy_tooltip (const char ∗ text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image ∗ deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image ∗ deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image & img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget ∗ widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget ∗ widget, void ∗ arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - "for hacks only.".
• void **position** (int X, int Y)
  Repositions the window or widget.

• void **redraw** ()
  Schedules the drawing of the widget.

• void **redraw_label** ()
  Schedules the drawing of the label.

• virtual void **resize** (int x, int y, int w, int h)
  Changes the size or position of the widget.

• Fl_Color **selection_color** () const
  Gets the selection color.

• void **selection_color** (Fl_Color a)
  Sets the selection color.

• void **set_active** ()
  Marks the widget as active without sending events or changing focus.

• void **set_changed** ()
  Marks the value of the widget as changed.

• void **set_output** ()
  Sets a widget to output only.

• void **set_visible** ()
  Makes the widget visible.

• void **set_visible_focus** ()
  Enables keyboard focus navigation with this widget.

• int **shortcut_label** () const
  Returns whether the widget's label uses '&' to indicate shortcuts.

• void **shortcut_label** (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

• virtual void **show** ()
  Makes a widget visible.

• void **size** (int W, int H)
  Changes the size of the widget.

• int **take_focus** ()
  Gives the widget the keyboard focus.

• unsigned int **takesevents** () const
  Returns if the widget is able to take events.

• int **test_shortcut** ()
  Returns true if the widget's label contains the entered 'x' shortcut.

• const char * **tooltip** () const
  Gets the current tooltip text.

• void **tooltip** (const char *text)
  Sets the current tooltip text.

• Fl_Window * **top_window** () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window * **top_window_offset** (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar **type** () const
  Gets the widget type.

• void **type** (uchar t)
  Sets the widget type.

• int **use_accents_menu** ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void * **user_data** () const
Gets the user data for this widget.

- `void user_data (Fl_Callback_User_Data ∗v, bool auto_free)`
  Sets the user data for this widget.

- `void user_data (void ∗v)`
  Sets the user data for this widget.

- `unsigned int visible () const`
  Returns whether a widget is visible.

- `unsigned int visible_focus () const`
  Checks whether this widget has a visible focus.

- `void visible_focus (int v)`
  Modifies keyboard focus navigation.

- `int visible_r () const`
  Returns whether a widget and all its parents are visible.

- `int w () const`
  Gets the widget width.

- `Fl_When when () const`
  Returns the conditions under which the callback is called.

- `void when (uchar i)`
  Sets the flags used to decide when a callback is called.

- `Fl_Window ∗window () const`
  Returns a pointer to the nearest parent window up the widget hierarchy.

- `int x () const`
  Gets the widget position in its window.

- `int y () const`
  Gets the widget position in its window.

- `virtual ~Fl_Widget ()`
  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- `static void default_callback (Fl_Widget ∗widget, void ∗data)`
  The default callback for all widgets that don't set a callback.

- `static unsigned int label_shortcut (const char ∗t)`
  Returns the Unicode value of the '&x' shortcut in a given text.

- `static int test_shortcut (const char ∗t, const bool require_alt=false)`
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- `enum { INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3, FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7, OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11, MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15, GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19, NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22, AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }
  flags possible values enumeration.
Protected Member Functions inherited from Fl_Counter

• void arrow_widths (int &w1, int &w2)
  Compute sizes (widths) of arrow boxes.

• void draw () FL_OVERRIDE
  Draws the widget.

Protected Member Functions inherited from Fl_Valuator

• Fl_Valuator (int X, int Y, int W, int H, const char ∗L)
  Creates a new Fl_Valuator widget using the given position, size, and label string.

• void handle_drag (double newvalue)
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.

• void handle_push ()
  Stores the current value in the previous value.

• void handle_release ()
  Called after an FL_WHEN_RELEASE event is received and before the callback.

• int horizontal () const
  Tells if the valuator is an FL_HORIZONTAL one.

• double previous_value () const
  Gets the previous floating point value before an event changed it.

• void set_value (double v)
  Sets the current floating point value.

• double softclamp (double)
  Clamps the value, but accepts v if the previous value is not already out of range.

• virtual void value_damage ()
  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.

• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• void draw_box () const
  Draws the widget box according its box style.

• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

• void draw_focus () const
  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

• void draw_label () const
  Draws the widget's label at the defined label position.

• void draw_label (int, int, int) const
  Draws the label in an arbitrary bounding box.

• Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.
12.130 Fl_Single_Window Class Reference

- **unsigned int flags () const**
  
  *Gets the widget flags mask.*

- **void h (int v)**
  
  *Internal use only.*

- **void set_flag (unsigned int c)**
  
  *Sets a flag in the flags mask.*

- **void w (int v)**
  
  *Internal use only.*

- **void x (int v)**
  
  *Internal use only.*

- **void y (int v)**
  
  *Internal use only.*

### 12.129.1 Detailed Description

This widget creates a counter with only 2 arrow buttons.

![Fl_Simple_Counter](image)

*Figure 12.43 Fl_Simple_Counter*

The documentation for this class was generated from the following files:

- Fl_Simple_Counter.H
- Fl_Counter.cxx

### 12.130 Fl_Single_Window Class Reference

This is the same as Fl_Window.

```c
#include <Fl_Single_Window.H>
```

Inheritance diagram for Fl_Single_Window:

```
Fl_Widget
   ↓
Fl_Group
   ↓
Fl_Window
   ↓
Fl_Single_Window
   ↓
Fl_Menu_Window
```

**Public Member Functions**

- **Fl_Single_Window (int W, int H, const char *l=0)**
  
  *Creates a new Fl_Single_Window widget using the given size, and label (title) string.*

- **Fl_Single_Window (int X, int Y, int W, int H, const char *l=0)**
  
  *Creates a new Fl_Single_Window widget using the given position, size, and label (title) string.*
• void show () FL_OVERRIDE
  Makes a widget visible.
• void show (int a, char **b)
  Same as Fl_Window::show(int a, char **b)

Public Member Functions inherited from Fl_Window

• virtual class Fl_Double_Window * as_double_window ()
  Return non-null if this is an Fl_Double_Window object.
• virtual class Fl_Overlay_Window * as_overlay_window ()
  Return non-null if this is an Fl_Overlay_Window object.
• Fl_Window const * as_window () const FL_OVERRIDE
• Fl_Window * as_window () FL_OVERRIDE
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• unsigned int border () const
  Returns whether the window possesses a border.
• void border (int b)
  Sets whether or not the window manager border is around the window.
• void clear_border ()
  Fast inline function to turn the window manager border off.
• void clear_modal_states ()
  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.
• void copy_label (const char * a)
  Sets the window titlebar label to a copy of a character string.
• void cursor (const Fl_RGB_Image *, int, int)
  Changes the cursor for this window using the provided image as cursor's shape.
• void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  For back compatibility only.
• void cursor (Fl_Cursor)
  Changes the cursor for this window.
• int decorated_h () const
  Returns the window height including any window title bar and any frame added by the window manager.
• int decorated_w () const
  Returns the window width including any frame added by the window manager.
• void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  For back compatibility only.
• void default_cursor (Fl_Cursor)
  Sets the default window cursor.
• void draw_backdrop ()
  Draw the background image if one is set and is aligned inside.
• Fl_Window (int w, int h, const char * title=0)
  Creates a window from the given width w, height h, and title.
• Fl_Window (int x, int y, int w, int h, const char *title=0)
  Creates a window from the given position (x, y), size (w, h) and title.
• void free_position ()
  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free
to position the window.
• void fullscreen ()
  Makes the window completely fill one or more screens, without any window manager border visible.
• unsigned int fullscreen_active () const
  Returns non zero if FULLSCREEN flag is set, 0 otherwise.
• void **fullscreen_off** ()
  Turns off any side effects of **fullscreen()**

• void **fullscreen_off** (int X, int Y, int W, int H)
  Turns off any side effects of **fullscreen()** and does **resize(x,y,w,h)**.

• void **fullscreen_screens** (int top, int bottom, int left, int right)
  Sets which screens should be used when this window is in fullscreen mode.

• int **handle** (int) FL_OVERRIDE
  Handles the specified event.

• void **hide** () FL_OVERRIDE
  Removes the window from the screen.

• void **hotspot** (const Fl_Widget &p, int offscreen=0)
  See void Fl_Window::**hotspot**(int x, int y, int offscreen = 0)

• void **hotspot** (const Fl_Widget *p, int offscreen=0)
  See void Fl_Window::**hotspot**(int x, int y, int offscreen = 0)

• void **hotspot** (int x, int y, int offscreen=0)
  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.

• const void * **icon** () const
  Gets the current icon window target dependent data.

• void **icon** (const Fl_RGB_Image *)
  Sets or resets a single window icon.

• void **icon** (const void *ic)
  Platform-specific method to set the window icon usable on Windows and X11 only.

• void **iconize** ()
  Iconifies the window.

• const char * **iconlabel** () const
  See void Fl_Window::**iconlabel**(const char *)

• void **iconlabel** (const char *)
  Sets the icon label.

• void **icons** (const Fl_RGB_Image *[], int)
  Sets the window icons.

• void **icons** (HICON big_icon, HICON small_icon)
  Sets the window icons using HICON handles (Windows platform only).

• const char * **label** () const
  See void Fl_Window::**label**(const char *)

• void **label** (const char *)
  Sets the window title bar label.

• void **label** (const char *label, const char *iconlabel)
  Sets the icon label.

• void **make_current** ()
  Sets things up so that the drawing functions in `<FL/fl_draw.H>` will go into this window.

• void **maximize** ()
  Maximizes a top-level window to its current screen.

• unsigned int **maximize_active** () const
  Returns whether the window is currently maximized.

• unsigned int **menu_window** () const
  Returns true if this window is a menu window.

• unsigned int **modal** () const
  Returns true if this window is modal.
• unsigned int **non_modal** () const
  
  Returns true if this window is modal or non-modal.

• **fl_uintptr_t os_id** ()
  
  Returns a platform-specific identification of a shown window, or 0 if not shown.

• unsigned int **override** () const
  
  Returns non zero if OVERRIDE flag is set, 0 otherwise.

• void **resize** (int X, int Y, int W, int H) **FL_OVERRIDE**
  
  Changes the size and position of the window.

• int **screen_num** ()
  
  The number of the screen containing the mapped window.

• void **screen_num** (int screen_num)
  
  Set the number of the screen where to map the window.

• void **set_menu_window** ()
  
  Marks the window as a menu window.

• void **set_modal** ()
  
  A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).

• void **set_non_modal** ()
  
  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.

• void **set_override** ()
  
  Activates the flags NOBORDER, OVERRIDE.

• void **set_tooltip_window** ()
  
  Marks the window as a tooltip window.

• const Fl_Image * **shape** ()
  
  Returns the image controlling the window shape or NULL.

• void **shape** (const Fl_Image &b)
  
  Set the window’s shape with an Fl_Image.

• void **shape** (const Fl_Image *img)
  
  Assigns a non-rectangular shape to the window.

• void **show** () **FL_OVERRIDE**
  
  Puts the window on the screen.

• void **show** (int argc, char **argv)
  
  Puts the window on the screen with show() and parses command-line arguments.

• int **shown** ()
  
  Returns non-zero if shown() has been called (but not hide() ).

• void **size_range** (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)
  
  Sets the allowable range the user can resize this window to.

• unsigned int **tooltip_window** () const
  
  Returns true if this window is a tooltip window.

• void **un_maximize** ()
  
  Returns a previously maximized top-level window to its previous size.

• void **wait_for_expose** ()
  
  Waits for the window to be displayed after calling show().

• int **x_root** () const
  
  Gets the x position of the window on the screen.

• const char * **xclass** () const
  
  Returns the xclass for this window, or a default.

• void **xclass** (const char *c)
  
  Sets the xclass for this window.

• int **y_root** () const
  
  Gets the y position of the window on the screen.

• virtual ~Fl_Window ()
  
  The destructor also deletes all the children.
Public Member Functions inherited from Fl_Group

- `Fl_Widget * _ddfdesign_kludge ()`
  - This is for forms compatibility only.
- `void add (Fl_Widget &)`
  - The widget is removed from its current group (if any) and then added to the end of this group.
- `void add (Fl_Widget *)`
  - See void Fl_Group::add(Fl_Widget &w)
- `void add_resizable (Fl_Widget &o)`
  - Adds a widget to the group and makes it the resizable widget.
- `Fl_Widget * const * array () const`
  - Returns a pointer to the array of children.
- `Fl_Group * as_group () const FL_OVERRIDE`
  - Returns an Fl_Group pointer if this widget is an Fl_Group.
- `void begin ()`
  - Sets the current group so you can build the widget tree by just constructing the widgets.
- `Fl_Widget * child (int n) const`
  - Returns array()[n].
- `int children () const`
  - Returns how many child widgets the group has.
- `void clear ()`
  - Deletes all child widgets from memory recursively.
- `unsigned int clip_children ()`
  - Returns the current clipping mode.
- `void clip_children (int c)`
  - Controls whether the group widget clips the drawing of child widgets to its bounding box.
- `virtual int delete_child (int n)`
  - Removes the widget at index from the group and deletes it.
- `void end ()`
  - Exactly the same as current(this->parent()).
- `int find (const Fl_Widget &o) const`
  - See int Fl_Group::find(const Fl_Widget *w) const.
- `int find (const Fl_Widget *) const`
  - Searches the child array for the widget and returns the index.
- `Fl_Group (int, int, int, int, const char * = 0)`
  - Creates a new Fl_Group widget using the given position, size, and label string.
- `void focus (Fl_Widget *)`
- `void forms_end ()`
  - This is for forms compatibility only.
- `void init_sizes ()`
  - Resets the internal array of widget sizes and positions.
- `void insert (Fl_Widget &, int i)`
  - The widget is removed from its current group (if any) and then inserted into this group.
- `void insert (Fl_Widget &, Fl_Widget *)`
  - This does insert(w, find(before)).
- `void remove (Fl_Widget &)`
  - Removes a widget from the group but does not delete it.
- `void remove (Fl_Widget *)`
  - Removes the widget o from the group.
- `void remove (int index)`
Removes the widget at index from the group but does not delete it.

- **Fl_Widget** + resizable () const
  Returns the group's resizable widget.

- **void** resizable (Fl_Widget &o)
  Sets the group's resizable widget.

- **void** resizable (Fl_Widget *o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- **virtual** ~Fl_Group ()
  The destructor also deletes all the children.

### Public Member Functions inherited from Fl_Widget

- **void** _clear_fullscreen ()
  Activates the widget.

- **void** _set_fullscreen ()
  Activates the widget.

- **unsigned int** active () const
  Returns whether the widget is active.

- **int** active_r () const
  Returns whether the widget and all of its parents are active.

- **Fl_Align** align () const
  Gets the label alignment.

- **void** align (Fl_Align alignment)
  Sets the label alignment.

- **long** argument () const
  Gets the current user data (long) argument that is passed to the callback function.

- **void** argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.

- **virtual class Fl_Gl_Window** + as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- **virtual class Fl_Gl_Window const** + as_gl_window () const

- **void** bind_deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

- **void** bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- **void** bind_image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.

- **void** bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype** box () const
  Gets the box type of the widget.

- **void** box (Fl_Boxtype new_box)
  Sets the box type for the widget.

- **Fl_Callback_p** callback () const
  Gets the current callback function for the widget.

- **void** callback (Fl_Callback *cb)
  Sets the current callback function for the widget.

- **void** callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

- **void** callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
  Checks if the widget value changed since the last callback.
- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()
  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)
  Clears or sets the damage flags.
- void clear_output ()
  Sets a widget to accept input.
- void clear_visible ()
  Hides the widget.
- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
- Fl_Color color () const
  Gets the background color of the widget.
- void color (Fl_Color bg)
  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
- Fl_Color color2 () const
  For back compatibility only.
- void color2 (unsigned a)
  For back compatibility only.
- int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.
- void copy_label (const char *new_label)
  Sets the current label.
- void copy_tooltip (const char *text)
  Sets the current tooltip text.
- uchar damage () const
  Returns non-zero if draw() needs to be called.
- void damage (uchar c)
  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
- int damage_resize (int, int, int, int)
  Internal use only.
- void deactivate ()
  Deactivates the widget.
- Fl_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)
Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const**
  
  Returns whether the inactive image is managed by the widget.

- **void do_callback (FL_Callback_Reason reason=FL_REASON_UNKNOWN)**
  
  Calls the widget callback function with default arguments.

- **void do_callback (FL_Widget *widget, long arg, FL_Callback_Reason reason=FL_REASON_UNKNOWN)**
  
  Calls the widget callback function with arbitrary arguments.

- **void do_callback (FL_Widget *widget, void *arg=0, FL_Callback_Reason reason=FL_REASON_UNKNOWN)**
  
  Calls the widget callback function with arbitrary arguments.

- **void draw_label (int, int, int, int, FL_Align) const**
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h () const**
  
  Gets the widget height.

- **FL_Image * image ()**
  
  Gets the image that is used as part of the widget label when in the active state.

- **const FL_Image * image () const**
  
  Gets the image that is used as part of the widget label when in the active state.

- **void image (FL_Image &img)**
  
  Sets the image to use as part of the widget label when in the active state.

- **void image (FL_Image *img)**
  
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const**
  
  Returns whether the image is managed by the widget.

- **int inside (const FL_Widget *wgt) const**
  
  Checks if this widget is a child of wgt.

- **int is_label_copied () const**
  
  Returns whether the current label was assigned with copy_label().

- **const char * label () const**
  
  Gets the current label text.

- **void label (const char *text)**
  
  Sets the current label pointer.

- **void label (FL_Labeltype a, const char *b)**
  
  Shortcut to set the label text and type in one call.

- **FL_Color labelcolor () const**
  
  Gets the label color.

- **void labelcolor (FL_Color c)**
  
  Sets the label color.

- **FL_Font labelfont () const**
  
  Gets the font to use.

- **void labelfont (FL_Font f)**
  
  Sets the font to use.

- **FL_Fontsize labelsize () const**
  
  Gets the font size in pixels.

- **void labelsize (FL_Fontsize pix)**
  
  Sets the font size in pixels.

- **FL_Labeltype labeltype () const**
  
  Gets the label type.

- **void labeltype (FL_Labeltype a)**
  
  Sets the label type.

- **void measure_label (int &ww, int &hh) const**
  
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group ∗ parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group ∗ p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int testShortcut ()
  Returns true if the widget's label contains the entered 'x' shortcut.
• const char ∗ tooltip () const
  Gets the current tooltip text.
• void tooltip (const char ∗ text)
  Sets the current tooltip text.
• Fl_Window ∗ top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  Checks whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Window

  • typedef struct HICON__ * HICON

Static Public Member Functions inherited from Fl_Window

  • static Fl_Window * current ()
    Returns the last window that was made current.
  • static void default_callback (Fl_Window *, void *v)
    Back compatibility: Sets the default callback v for win to call on close event.
  • static void default_icon (const Fl_RGB_Image *)
    Sets a single default window icon.
  • static void default_icons (const Fl_RGB_Image *[], int)
    Sets the default window icons.
  • static void default_icons (HICON big_icon, HICON small_icon)
    Sets the default window icons (Windows platform only).
**static const char ∗ default_xclass ()**  
Returns the default xclass.

**static void default_xclass (const char ∗)  
Sets the default window xclass.**

**static bool is_a_rescale ()**  
Returns true when a window is being rescaled.

**static char show_next_window_iconic ()**  
Returns the static flag whether the next window should be opened iconified.

**static void show_next_window_iconic (char stat)**  
Sets a static flag whether the next window should be opened iconified.

### Static Public Member Functions inherited from Fl_Group

- **static Fl_Group ∗ current ()**  
  Returns the currently active group.

- **static void current (Fl_Group ∗g)**  
  Sets the current group.

### Static Public Member Functions inherited from Fl_Widget

- **static void default_callback (Fl_Widget ∗widget, void ∗data)**  
The default callback for all widgets that don’t set a callback.

- **static unsigned int label_shortcut (const char ∗t)**  
  Returns the Unicode value of the '&x' shortcut in a given text.

- **static int test_shortcut (const char ∗, const bool require_alt=false)**  
  Returns true if the given text t contains the entered '&x' shortcut.

### Protected Types inherited from Fl_Widget

- **enum {**
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }**
  flags possible values enumeration.

### Protected Member Functions inherited from Fl_Window

- **void default_size_range ()**  
  Protected method to calculate the default size range of a window.

- **void draw () FL_OVERRIDE**  
  Draws the widget.

- **virtual void flush ()**  
  Forces the window to be drawn, this window is also made current and calls draw().

- **int force_position () const**  
  Returns the internal state of the window's FORCE_POSITION flag.
• void force_position (int force)
  Sets an internal flag that tells FLTK and the window manager to honor position requests.

• void free_icons ()
  Deletes all icons previously attached to the window.

• int is_resizable ()
  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

• Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.

• void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.

• void draw_children ()
  Draws all children of the group.

• void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.

• virtual int on_insert (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.

• virtual int on_move (int, int)
  Allow derived groups to act when a widget is moved within the group.

• virtual void on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.

• int * sizes ()
  Returns the internal array of widget sizes and positions.

• void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.

• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• void draw_box () const
  Draws the widget box according its box style.

• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

• void draw_focus () const
  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

• void draw_label () const
  Draws the widget's label at the defined label position.

• void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.

• Fl_Widget (int x, int y, int w, int h, const char *label=0L)
Creates a widget at the given position and size.

- unsigned int flags () const
  Gets the widget flags mask.
- void h (int v)
  Internal use only.
- void set_flag (unsigned int c)
  Sets a flag in the flags mask.
- void w (int v)
  Internal use only.
- void x (int v)
  Internal use only.
- void y (int v)
  Internal use only.

Static Protected Attributes inherited from Fl_Window

- static Fl_Window *current_
  Stores the last window that was made current.

12.130.1 Detailed Description

This is the same as Fl_Window. However, it is possible that some implementations will provide double-buffered windows by default. This subclass can be used to force single-buffering. This may be useful for modifying existing programs that use incremental update, or for some types of image data, such as a movie flipbook.

12.130.2 Member Function Documentation

12.130.2.1 show()

void Fl_Single_Window::show ( ) [virtual]

Makes a widget visible.
An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other events like FL_SHOW.
The visible() method returns true if the widget is set to be visible. The visible_r() method returns true if the widget and all of its parents are visible. A widget is only visible if visible() is true on it and all of its parents.
Changing it will send FL_SHOW or FL_HIDE events to the widget. Do not change it if the parent is not visible, as this will send false FL_SHOW or FL_HIDE events to the widget. redraw() is called if necessary on this or the parent.

See also

hide(), visible(), visible_r()

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

- Fl_Single_Window.H
- Fl_Single_Window.cxx

12.131 Fl_Slider Class Reference

The Fl_Slider widget contains a sliding knob inside a box.
#include <Fl_Slider.H>

Inheritance diagram for Fl_Slider:
Public Member Functions

- void **bounds** (double a, double b)
  
  *Sets the minimum (a) and maximum (b) values for the valuator widget.*

- **FL_Slider** (int X, int Y, int W, int H, const char ∗L=0)
  
  *Creates a new Fl_Slider widget using the given position, size, and label string.*

- **FL_Slider** (uchar t, int X, int Y, int W, int H, const char ∗L)
  
  *Creates a new Fl_Slider widget using the given type, position, size, and label string.*

- int **handle** (int) FL_OVERRIDE
  
  *Handles the specified event.*

- int **scrollvalue** (int pos, int size, int first, int total)
  
  *Sets the size and position of the sliding knob in the box.*

- **FL_Boxtype** **slider** () const
  
  *Gets the slider box type.*

- void **slider** (FL_Boxtype c)
  
  *Sets the slider box type.*

- float **slider_size** () const
  
  *Get the dimensions of the moving piece of slider.*

- void **slider_size** (double v)
  
  *Set the dimensions of the moving piece of slider.*

Public Member Functions inherited from **FL_Valuator**

- void **bounds** (double a, double b)
  
  *Sets the minimum (a) and maximum (b) values for the valuator widget.*

- double **clamp** (double)
  
  *Clamps the passed value to the valuator range.*

- virtual int **format** (char ∗)
  
  *Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.*

- double **increment** (double, int)
  
  *Adds n times the step value to the passed value.*

- double **maximum** () const
  
  *Gets the maximum value for the valuator.*

- void **maximum** (double a)
  
  *Sets the maximum value for the valuator.*

- double **minimum** () const
  
  *Gets the minimum value for the valuator.*

- void **minimum** (double a)
  
  *Sets the minimum value for the valuator.*

- void **precision** (int digits)
  
  *Sets the step value to 1.0 / 10^{digits}.*

- void **range** (double a, double b)
  
  *Sets the minimum and maximum values for the valuator.*

- double **round** (double)
Round the passed value to the nearest step increment.

- double step() const
  Gets or sets the step value.

- void step(double a, int b)
  See double Fl_Valuator::step() const

- void step(double s)
  See double Fl_Valuator::step() const.

- void step(int a)
  See double Fl_Valuator::step() const

- double value() const
  Gets the floating point(double) value.

- int value(double)
  Sets the current value.

Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen()
- void _set_fullscreen()
- void activate()
  Activates the widget.
- unsigned int active() const
  Returns whether the widget is active.
- int active_r() const
  Returns whether the widget and all of its parents are active.
- Fl_Align align() const
  Gets the label alignment.
- void align(Fl_Align alignment)
  Sets the label alignment.
- long argument() const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument(long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * as_gl_window() const
- virtual Fl_Group * as_group()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * as_group() const
- virtual Fl_Window * as_window()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * as_window() const
- void bind_deimage(Fl_Image * img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage(int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image(Fl_Image * img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image(int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box() const
Gets the box type of the widget.

- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback +cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback +cb, Fl_Callback_User_Data +p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback +cb, void +p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 +cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 +cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
  Checks if the widget value changed since the last callback.
- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()
  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)
  Clears or sets the damage flags.
- void clear_output ()
  Sets a widget to accept input.
- void clear_visible ()
  Hides the widget.
- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
- Fl_Color color () const
  Gets the background color of the widget.
- void color (Fl_Color bg)
  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
- Fl_Color color2 () const
  For back compatibility only.
- void color2 (unsigned a)
  For back compatibility only.
- int contains (const Fl_Widget +w) const
  Checks if w is a child of this widget.
- void copy_label (const char +new_label)
  Sets the current label.
- void copy_tooltip (const char +text)
  Sets the current tooltip text.
- uchar damage () const
  Returns non-zero if draw() needs to be called.
- void damage (uchar c)
  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int **damage_resize**(int, int, int, int)
  
  **Internal use only.**

• void **deactivate**()
  
  Deactivates the widget.

• Fl_Img**age** **+** deimage()
  
  **Gets the image that is used as part of the widget label when in the inactive state.**

• const Fl_Img**age** **+** deimage() const
  
  **Gets the image that is used as part of the widget label when in the inactive state.**

• void **deimage**(Fl_Img**age** &img)
  
  **Sets the image to use as part of the widget label when in the inactive state.**

• void **deimage**(Fl_Img**age** *img)
  
  **Sets the image to use as part of the widget label when in the inactive state.**

• int **deimage_bound**() const
  
  **Returns whether the inactive image is managed by the widget.**

• void **do_callback**(Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with default arguments.

• void **do_callback**(Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with arbitrary arguments.

• void **do_callback**(Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with arbitrary arguments.

• void **draw_label**(int, int, int, int, Fl_Align) const
  
  **Draws the label in an arbitrary bounding box with an arbitrary alignment.**

• int **h**() const
  
  **Gets the widget height.**

• virtual void **hide**()
  
  **Makes a widget invisible.**

• Fl_Img**age** **+** image()
  
  **Gets the image that is used as part of the widget label when in the active state.**

• const Fl_Img**age** **+** image() const
  
  **Gets the image that is used as part of the widget label when in the active state.**

• void **image**(Fl_Img**age** &img)
  
  **Sets the image to use as part of the widget label when in the active state.**

• void **image**(Fl_Img**age** *img)
  
  **Sets the image to use as part of the widget label when in the active state.**

• int **image_bound**() const
  
  **Returns whether the image is managed by the widget.**

• int **inside**(const Fl_Widget *wgt) const
  
  **Checks if this widget is a child of wgt.**

• int **is_label_copied**() const
  
  **Returns whether the current label was assigned with copy_label().**

• const char **+** label() const
  
  **Gets the current label text.**

• void **label**(const char **+**text)
  
  **Sets the current label pointer.**

• void **label**(Fl_Labeltype a, const char **+**b)
  
  **Shortcut to set the label text and type in one call.**

• Fl_Color **labelcolor**() const
  
  **Gets the label color.**

• void **labelcolor**(Fl_Color c)
  
  **Sets the label color.**

• Fl_Font **labelfont**() const
Gets the font to use.

- void `labelfont (Fl_Font f)`
  
  Sets the font to use.

- `Fl_Fontsize` `labelfsize` () const
  
  Gets the font size in pixels.

- void `labelfsize (Fl_Fontsize pix)`
  
  Sets the font size in pixels.

- `Fl_Labeltype` `labelftype` () const
  
  Gets the label type.

- void `labelftype (Fl_Labeltype a)`
  
  Sets the label type.

- void `measure_label (int &ww, int &hh)` const
  
  Sets width `ww` and height `hh` accordingly with the label size.

- bool `needs_keyboard` () const
  
  Returns whether this widget needs a keyboard.

- void `needs_keyboard (bool needs)`
  
  Sets whether this widget needs a keyboard.

- unsigned int `output` () const
  
  Returns if a widget is used for output only.

- `Fl_Group` * `parent` () const
  
  Returns a pointer to the parent widget.

- void `parent (Fl_Group *p)`
  
  Internal use only - "for hacks only".

- void `position (int X, int Y)`
  
  Repositions the window or widget.

- void `redraw ()`
  
  Schedules the drawing of the widget.

- void `redraw_label ()`
  
  Schedules the drawing of the label.

- virtual void `resize (int x, int y, int w, int h)`
  
  Changes the size or position of the widget.

- `Fl_Color` `selection_color` () const
  
  Gets the selection color.

- void `selection_color (Fl_Color a)`
  
  Sets the selection color.

- void `set_active ()`
  
  Marks the widget as active without sending events or changing focus.

- void `set_changed ()`
  
  Marks the value of the widget as changed.

- void `set_output ()`
  
  Sets a widget to output only.

- void `set_visible ()`
  
  Makes the widget visible.

- void `set_visible_focus ()`
  
  Enables keyboard focus navigation with this widget.

- int `shortcut_label` () const
  
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.

- void `shortcut_label (int value)`
  
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.

- virtual void `show ()`
  
  Makes a widget visible.
• void **size** (int W, int H)
  Changes the size of the widget.

• int **take_focus** ()
  Gives the widget the keyboard focus.

• unsigned int **takeevents** () const
  Returns if the widget is able to take events.

• int **test_shortcut** ()
  Returns true if the widget's label contains the entered '&x' shortcut.

• const char ∗ **tooltip** () const
  Gets the current tooltip text.

• void **tooltip** (const char ∗text)
  Sets the current tooltip text.

• **Fl_Window** ∗ **top_window** () const
  Returns a pointer to the top-level window for the widget.

• **Fl_Window** ∗ **top_window_offset** (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar **type** () const
  Gets the widget type.

• void **type** (uchar t)
  Sets the widget type.

• int **use_accents_menu** ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void ∗ **user_data** () const
  Gets the user data for this widget.

• void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.

• void **user_data** (void ∗v)
  Sets the user data for this widget.

• unsigned int **visible** () const
  Returns whether a widget is visible.

• unsigned int **visible_focus** () const
  Checks whether this widget has a visible focus.

• void **visible_focus** (int v)
  Modifies keyboard focus navigation.

• int **visible_r** () const
  Returns whether a widget and all its parents are visible.

• int **w** () const
  Gets the widget width.

• **Fl_When** **when** () const
  Returns the conditions under which the callback is called.

• void **when** (uchar i)
  Sets the flags used to decide when a callback is called.

• **Fl_Window** ∗ **window** () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int **x** () const
  Gets the widget position in its window.

• int **y** () const
  Gets the widget position in its window.

• virtual ∼Fl_Widget ()
  Destroys the widget.
Protected Member Functions

- void **draw () **FL_OVERRIDE
  Draws the widget.
- void **draw**(int, int, int, int)
- int **handle**(int, int, int, int)

Protected Member Functions inherited from **Fl_Valuator**

- **Fl_Valuator**(int X, int Y, int W, int H, const char ∗L)
  Creates a new **Fl_Valuator** widget using the given position, size, and label string.
- void **handle_drag**(double newvalue)
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push()**
  Stores the current value in the previous value.
- void **handle_release()**
  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal ()** const
  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value ()** const
  Gets the previous floating point value before an event changed it.
- void **set_value**(double v)
  Sets the current floating point value.
- double **softclamp**(double)
  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void **value_damage ()**
  Asks for partial redraw.

Protected Member Functions inherited from **Fl_Widget**

- void **clear_flag**(unsigned int c)
  Clears a flag in the flags mask.
- void **draw_backdrop ()** const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box ()** const
  Draws the widget box according its box style.
- void **draw_box**(Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box**(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void **draw_focus ()** const
  Draws a focus rectangle around the widget.
- void **draw_focus**(Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
- void **draw_focus**(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void **draw_label ()** const
  Draws the widget's label at the defined label position.
- void **draw_label**(int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- **Fl_Widget**(int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.
• unsigned int **flags** () const
  
  *Gets the widget flags mask.*

• void **h** (int v)
  
  *Internal use only.*

• void **set_flag** (unsigned int c)
  
  *Sets a flag in the flags mask.*

• void **w** (int v)
  
  *Internal use only.*

• void **x** (int v)
  
  *Internal use only.*

• void **y** (int v)
  
  *Internal use only.*

### Additional Inherited Members

### Static Public Member Functions inherited from Fl_Widget

• static void **default_callback** (Fl_Widget *widget, void *data)
  
  *The default callback for all widgets that don't set a callback.*

• static unsigned int **label_shortcut** (const char *t)
  
  *Returns the Unicode value of the '&x' shortcut in a given text.*

• static int **test_shortcut** (const char *t, const bool require_alt=false)
  
  *Returns true if the given text t contains the entered '&x' shortcut.*

### Protected Types inherited from Fl_Widget

• enum {
  
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , Copied_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  Menu_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , Copied_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCELETS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }

  *flags possible values enumeration.*

### Detailed Description

The Fl_Slider widget contains a sliding knob inside a box. It is often used as a scrollbar. Moving the box all the way to the top/left sets it to the minimum(), and to the bottom/right to the maximum(). The minimum() may be greater than the maximum() to reverse the slider direction. Use void Fl_Widget::type(int) to set how the slider is drawn, which can be one of the following:

• FL_VERTICAL - Draws a vertical slider (this is the default).

• FL_HORIZONTAL - Draws a horizontal slider.

• FL_VERT_FILL_SLIDER - Draws a filled vertical slider, useful as a progress or value meter.

• FL_HOR_FILL_SLIDER - Draws a filled horizontal slider, useful as a progress or value meter.

• FL_VERT_NICE_SLIDER - Draws a vertical slider with a nice looking control knob.
• FL_HOR_NICE_SLIDER - Draws a horizontal slider with a nice looking control knob.

![Figure 12.44 Fl_Slider](image)

### 12.131.2 Constructor & Destructor Documentation

#### 12.131.2.1 Fl_Slider()

```cpp
Fl_Slider::Fl_Slider ( 
    int X,
    int Y,
    int W,
    int H,
    const char ∗ L = 0 )
```

Creates a new Fl_Slider widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX.

### 12.131.3 Member Function Documentation

#### 12.131.3.1 bounds()

```cpp
void Fl_Slider::bounds ( 
    double a,
    double b )
```

Sets the minimum (a) and maximum (b) values for the valuator widget. If at least one of the values is changed, a partial redraw is asked.

#### 12.131.3.2 draw()

```cpp
void Fl_Slider::draw ( ) [protected], [virtual]
```

Draws the widget. Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead. Override this function to draw your own widgets. If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget ∗ s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget. Reimplemented in Fl_Value_Slider.

#### 12.131.3.3 handle()

```cpp
int Fl_Slider::handle ( 
    int event ) [virtual]
```

Handles the specified event. You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval. One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in  | event | the kind of event received |

Return values

| 0   | if the event was not used or understood |
| 1   | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Widget.
Reimplemented in Fl_Value_Slider.

12.131.3.4 scrollvalue()

int Fl_Slider::scrollvalue (int pos, int size, int first, int total)

Sets the size and position of the sliding knob in the box.

Parameters

| in  | pos | position of first line displayed |
| in  | size | size of window in lines |
| in  | first | number of first line |
| in  | total | total number of lines Returns Fl_Valuator::value(p) |

12.131.3.5 slider_size()

void Fl_Slider::slider_size (double v)

Set the dimensions of the moving piece of slider.
This is the fraction of the size of the entire widget. If you set this to 1 then the slider cannot move. The default value is .08.
For the "fill" sliders this is the size of the area around the end that causes a drag effect rather than causing the slider to jump to the mouse.
The documentation for this class was generated from the following files:

- Fl_Slider.H
- Fl_Slider.cxx

12.132 Fl_Spinner Class Reference

This widget is a combination of a numerical input widget and repeat buttons.
#include <Fl_Spinner.H>

Inheritance diagram for Fl_Spinner:

```
Fl_Widget
   |
   v
Fl_Group
   |
   v
Fl_Spinner
```

Classes

- class Fl_Spinner_Input

Public Member Functions

- Fl_Color color() const
  Returns the background color of the spinner widget's input field.

- void color(Fl_Color v)
  Sets the background color of the spinner widget's input field.

- Fl_Spinner(int X, int Y, int W, int H, const char *L=0)
  Creates a new Fl_Spinner widget using the given position, size, and label string.

- const char * format() const
  Returns the format string for the value.

- void format(const char *f)
  Sets the format string for the value.

- int handle(int event) FL_OVERRIDE
  Handles the specified event.

- double maximum() const
  Gets the maximum value of the widget.

- void maximum(double m)
  Sets the maximum value of the widget.

- int maximum_size() const
  Returns the maximum width of the input field.

- void maximum_size(int m)
  Sets the maximum width of the input field.

- double minimum() const
  Gets the minimum value of the widget.

- void minimum(double m)
  Sets the minimum value of the widget.

- void range(double a, double b)
  Sets the minimum and maximum values for the widget.

- void resize(int X, int Y, int W, int H) FL_OVERRIDE
  Resizes the Fl_Group widget and all of its children.

- Fl_Color selection_color() const
  Returns the selection color of the spinner widget's input field.

- void selection_color(Fl_Color val)
  Sets the selection color of the spinner widget's input field.

- double step() const
  Gets the amount to change the value when the user clicks a button.

- void step(double s)
Sets or returns the amount to change the value when the user clicks a button.

* `Fl_Color textcolor() const`
  Gets the color of the text in the input field.

* `void textcolor(Fl_Color c)`
  Sets the color of the text in the input field.

* `Fl_Font textfont() const`
  Gets the font of the text in the input field.

* `void textfont(Fl_Font f)`
  Sets the font of the text in the input field.

* `Fl_Fontsize textsize() const`
  Gets the size of the text in the input field.

* `void textsize(Fl_Fontsize s)`
  Sets the size of the text in the input field.

* `uchar type() const`
  Gets the numeric representation in the input field.

* `void type(uchar v)`
  Sets the numeric representation in the input field.

* `double value() const`
  Gets the current value of the widget.

* `void value(double v)`
  Sets the current value of the input widget.

* `int wrap() const`
  Gets the wrap mode of the `Fl_Spinner` widget.

* `void wrap(int set)`
  Sets whether the spinner wraps around at upper and lower bounds.

### Public Member Functions inherited from `Fl_Group`

* `FL_Widget *&_ddfdesign_kludge()`
  This is for forms compatibility only.

* `void add(Fl_Widget &)`
  The widget is removed from its current group (if any) and then added to the end of this group.

* `void add(Fl_Widget *o)`
  See void `Fl_Group::add(Fl_Widget &w)`

* `void add_resizable(Fl_Widget &o)`
  Adds a widget to the group and makes it the resizable widget.

* `FL_Widget *const * array()` const
  Returns a pointer to the array of children.

* `Fl_Group const * as_group() const FL_OVERRIDE`
  Returns an `Fl_Group` pointer if this widget is an `Fl_Group`.

* `void begin()`
  Sets the current group so you can build the widget tree by just constructing the widgets.

* `FL_Widget * child(int n)` const
  Returns `array()[n]`.

* `int children()` const
  Returns how many child widgets the group has.

* `void clear()`
  Deletes all child widgets from memory recursively.

* `unsigned int clip_children()`
  Returns the current clipping mode.
• void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

• virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.

• void end ()
  Exactly the same as current(this->parent());

• int find (const Fl_Widget &o) const
  See int Fl_Group::find(const Fl_Widget *w) const.

• int find (const Fl_Widget *) const
  Searches the child array for the widget and returns the index.

• Fl_Group (int, int, int, int, const char *)=0
  Creates a new Fl_Group widget using the given position, size, and label string.

• void focus (Fl_Widget *)

• void forms_end ()
  This is for forms compatibility only.

• void init_sizes ()
  Resets the internal array of widget sizes and positions.

• void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.

• void insert (Fl_Widget &o, Fl_Widget *before)
  This does insert(w, find(before)).

• void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.

• void remove (Fl_Widget *o)
  Removes the widget o from the group.

• void remove (int index)
  Removes the widget at index from the group but does not delete it.

• Fl_Widget *resizable () const
  Returns the group's resizable widget.

• void resizable (Fl_Widget &o)
  Sets the group's resizable widget.

• void resizable (Fl_Widget *o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

• virtual ~Fl_Group ()
  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()

• void _set_fullscreen ()

• void activate ()
  Activates the widget.

• unsigned int active () const
  Returns whether the widget is active.

• int active_r () const
  Returns whether the widget and all of its parents are active.

• Fl_Align align () const
  Gets the label alignment.

• void align (Fl_Align alignment)
  Sets the label alignment.

• long argument () const

Generated by Doxygen
Gets the current user data (long) argument that is passed to the callback function.

- **void argument** (long v)
  
  Sets the current user data (long) argument that is passed to the callback function.

- **virtual class Fl_Gl_Window** ∗ as_gl_window ()
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- **virtual class Fl_Gl_Window const ∗ as_gl_window ()** const
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- **virtual Fl_Window** ∗ as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- **virtual Fl_Window const ∗ as_window ()** const
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- **void bind_deimage (Fl_Image ∗ img)**
  
  Sets the image to use as part of the widget label when in the inactive state.

- **void bind_deimage (int f)**
  
  Sets the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- **void bind_image (Fl_Image ∗ img)**
  
  Sets the image to use as part of the widget label when in the active state.

- **void bind_image (int f)**
  
  Sets the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype box () const**
  
  Gets the box type of the widget.

- **void box (Fl_Boxtype new_box)**
  
  Sets the box type for the widget.

- **Fl_Callback_p callback () const**
  
  Gets the current callback function for the widget.

- **void callback (Fl_Callback ∗ cb)**
  
  Sets the current callback function for the widget.

- **void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)**
  
  Sets the current callback function and managed user data for the widget.

- **void callback (Fl_Callback ∗ cb, void ∗ p)**
  
  Sets the current callback function and data for the widget.

- **void callback (Fl_Callback0 ∗ cb)**
  
  Sets the current callback function for the widget.

- **void callback (Fl_Callback1 ∗ cb, long p=0)**
  
  Sets the current callback function for the widget.

- **unsigned int changed () const**
  
  Checks if the widget value changed since the last callback.

- **void clear_active ()**
  
  Marks the widget as inactive without sending events or changing focus.

- **void clear_changed ()**
  
  Marks the value of the widget as unchanged.

- **void clear_damage (uchar c=0)**
  
  Clears or sets the damage flags.

- **void clear_output ()**
  
  Sets a widget to accept input.

- **void clear_visible ()**
  
  Hides the widget.

- **void clear_visible_focus ()**
  
  Disables keyboard focus navigation with this widget.

- **Fl_Color color () const**
  
  Gets the background color of the widget.

- **void color (Fl_Color bg)**
  
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

• Fl_Color color2 () const
  For back compatibility only.

• void color2 (unsigned a)
  For back compatibility only.

• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.

• void copy_label (const char *new_label)
  Sets the current label.

• void copy_tooltip (const char *text)
  Sets the current tooltip text.

• uchar damage () const
  Returns non-zero if draw() needs to be called.

• void damage (uchar c)
  Sets the damage bits for the widget.

• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

• int damage_resize (int, int, int, int)
  Internal use only.

• void deactivate ()
  Deactivates the widget.

• Fl_IMAGE = deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

• const Fl_IMAGE = deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

• void deimage (Fl_IMAGE &img)
  Sets the image to use as part of the widget label when in the inactive state.

• void deimage (Fl_IMAGE *img)
  Sets the image to use as part of the widget label when in the inactive state.

• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• int h () const
  Gets the widget height.

• virtual void hide ()
  Makes a widget invisible.

• Fl_IMAGE = image ()
  Gets the image that is used as part of the widget label when in the active state.

• const Fl_IMAGE = image () const
  Gets the image that is used as part of the widget label when in the active state.

• void image (Fl_IMAGE &img)
  Sets the image to use as part of the widget label when in the active state.

• void image (Fl_IMAGE *img)
Sets the image to use as part of the widget label when in the active state.

- `int image_bound () const`
  Returns whether the image is managed by the widget.

- `int inside (const Fl_Widget *wgt) const`
  Checks if this widget is a child of `wgt`.

- `int is_label_copied () const`
  Returns whether the current label was assigned with `copy_label()`.

- `const char * label () const`
  Gets the current label text.

- `void label (const char *text)`
  Sets the current label pointer.

- `void label (Fl_Labeltype a, const char *b)`
  Shortcut to set the label text and type in one call.

- `Fl_Color labelcolor () const`
  Gets the label color.

- `void labelcolor (Fl_Color c)`
  Sets the label color.

- `Fl_Font labelfont () const`
  Gets the font to use.

- `void labelfont (Fl_Font f)`
  Sets the font to use.

- `Fl_Fontsize labelsize () const`
  Gets the font size in pixels.

- `void labelsize (Fl_Fontsize pix)`
  Sets the font size in pixels.

- `Fl_Labeltype labeltype () const`
  Gets the label type.

- `void labeltype (Fl_Labeltype a)`
  Sets the label type.

- `void measure_label (int &ww, int &hh) const`
  Sets width `ww` and height `hh` accordingly with the label size.

- `bool needs_keyboard () const`
  Returns whether this widget needs a keyboard.

- `void needs_keyboard (bool needs)`
  Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  Returns if a widget is used for output only.

- `Fl_Group * parent () const`
  Returns a pointer to the parent widget.

- `void parent (Fl_Group *p)`
  Internal use only - “for hacks only”.

- `void position (int X, int Y)`
  Repositions the window or widget.

- `void redraw ()`
  Schedules the drawing of the widget.

- `void redraw_label ()`
  Schedules the drawing of the label.

- `Fl_Color selection_color () const`
  Gets the selection color.

- `void selection_color (Fl_Color a)`
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget’s label contains the entered ‘&x’ shortcut.
• const char ∗ tooltip () const
  Gets the current tooltip text.
• void tooltip (const char ∗text)
  Sets the current tooltip text.
• Fl_Window ∗ top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void ∗ user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void ∗v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
Returns whether a widget and all its parents are visible.

- **int w () const**
  Gets the widget width.

- **Fl_When when () const**
  Returns the conditions under which the callback is called.

- **void when (uchar i)**
  Sets the flags used to decide when a callback is called.

- **Fl_Window * window () const**
  Returns a pointer to the nearest parent window up the widget hierarchy.

- **int x () const**
  Gets the widget position in its window.

- **int y () const**
  Gets the widget position in its window.

- **virtual ~Fl_Widget ()**
  Destroys the widget.

### Protected Member Functions

- **void draw () FL_OVERRIDE**
  Draws the widget.

### Protected Member Functions inherited from Fl_Group

- **Fl_Rect * bounds ()**
  Returns the internal array of widget sizes and positions.

- **void draw_child (Fl_Widget &widget) const**
  Forces a child to redraw.

- **void draw_children ()**
  Draws all children of the group.

- **void draw_outside_label (const Fl_Widget &widget) const**
  Parents normally call this to draw outside labels of child widgets.

- **virtual int on_insert (Fl_Widget *, int)**
  Allow derived groups to act when a widget is added as a child.

- **virtual int on_move (int, int)**
  Allow derived groups to act when a widget is moved within the group.

- **virtual void on_remove (int)**
  Allow derived groups to act when a child widget is removed from the group.

- **int * sizes ()**
  Returns the internal array of widget sizes and positions.

- **void update_child (Fl_Widget &widget) const**
  Draws a child only if it needs it.

### Protected Member Functions inherited from Fl_Widget

- **void clear_flag (unsigned int c)**
  Clears a flag in the flags mask.

- **void draw_backdrop () const**
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- **void draw_box () const**
  Draws the widget box according its box style.

- **void draw_box (Fl_Boxtype t, Fl_Color c) const**
  Draws a box of type t, of color c at the widget's position and size.
• void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type t, of color c at the position X,Y and size W,H.

• void **draw_focus** () const
  
  Draws a focus rectangle around the widget.

• void **draw_focus** (Fl_Boxtype t, int X, int Y, int W, int H) const
  
  Draws a focus rectangle around the widget.

• void **draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.

• void **draw_label** () const
  
  Draws the widget's label at the defined label position.

• void **draw_label** (int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.

• **Fl_Widget** (int x, int y, int w, int h, const char ∗label=0L)
  
  Creates a widget at the given position and size.

• unsigned int **flags** () const
  
  Gets the widget flags mask.

• void **h** (int v)
  
  Internal use only.

• void **set_flag** (unsigned int c)
  
  Sets a flag in the flags mask.

• void **w** (int v)
  
  Internal use only.

• void **x** (int v)
  
  Internal use only.

• void **y** (int v)
  
  Internal use only.

**Protected Attributes**

  • Fl_Repeat_Button down_button_

  • Fl_Spinner_Input input_

  • Fl_Repeat_Button up_button_

**Additional Inherited Members**

**Static Public Member Functions inherited from Fl_Group**

  • static Fl_Group ∗current ()
    
    Returns the currently active group.

  • static void current (Fl_Group ∗g)
    
    Sets the current group.

**Static Public Member Functions inherited from Fl_Widget**

  • static void default_callback (Fl_Widget ∗widget, void ∗data)
    
    The default callback for all widgets that don't set a callback.

  • static unsigned int label_shortcut (const char ∗t)
    
    Returns the Unicode value of the '&x' shortcut in a given text.

  • static int test_shortcut (const char ∗t, const bool require_alt=false)
    
    Returns true if the given text t contains the entered '&x' shortcut.
Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9,copied_label = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTOP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

12.132.1 Detailed Description

This widget is a combination of a numerical input widget and repeat buttons.
The user can either type into the input area or use the buttons to change the value.

![Figure 12.45 Fl_Spinner widget](image)

12.132.2 Constructor & Destructor Documentation

12.132.2.1 Fl_Spinner()

Fl_Spinner::Fl_Spinner ( int X, int Y, int W, int H, const char ∗ L = 0 )

Creates a new Fl_Spinner widget using the given position, size, and label string.
The inherited destructor destroys the widget and any value associated with it.

12.132.3 Member Function Documentation

12.132.3.1 draw()

void Fl_Spinner::draw ( ) [protected], [virtual]

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as
soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded
scrollbar, you can do it (because draw() is virtual) like this:

```c++
Fl_Widget ∗ s = ∗ scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Group.

Generated by Doxygen
12.132.3.2 handle()

```cpp
int Fl_Spinner::handle (int event) [virtual]
```
Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in event | the kind of event received |

Return values

| 0 | if the event was not used or understood |
| 1 | if the event was used and can be deleted |

See also

- `Fl_Event`

Reimplemented from `Fl_Group`.

12.132.3.3 resize()

```cpp
void Fl_Spinner::resize (int X, int Y, int W, int H) [virtual]
```
Resizes the Fl_Group widget and all of its children.
The Fl_Group widget first resizes itself, and then it moves and resizes all its children according to the rules documented for Fl_Group::resizable(Fl_Widget*)

See also

- `Fl_Group::resizable(Fl_Widget*)`
- `Fl_Group::resizable()`
- `Fl_Widget::resize(int,int,int,int)`

Reimplemented from `Fl_Group`.

12.132.3.4 step()[1/2]

```cpp
double Fl_Spinner::step () const [inline]
```
Gets the amount to change the value when the user clicks a button.

See also

- `Fl_Spinner::step(double)`
12.132.3.5 step() [2/2]

void Fl_Spinner::step (double s)

Sets or returns the amount to change the value when the user clicks a button. Before setting step to a non-integer value, the spinner type() should be changed to floating point.

See also

double Fl_Spinner::step() const

12.132.3.6 type() [1/2]

uchar Fl_Spinner::type () const [inline]

Gets the numeric representation in the input field.

See also

Fl_Spinner::type(uchar)

12.132.3.7 type() [2/2]

void Fl_Spinner::type (uchar v)

Sets the numeric representation in the input field. Valid values are FL_INT_INPUT and FL_FLOAT_INPUT. Also changes the format() template. Setting a new spinner type via a superclass pointer will not work.

Note

type() is not a virtual function.

12.132.3.8 value()

void Fl_Spinner::value (double v) [inline]

Sets the current value of the input widget. Before setting value to a non-integer value, the spinner type() should be changed to floating point.

12.132.3.9 wrap() [1/2]

int Fl_Spinner::wrap ( ) const [inline]

Gets the wrap mode of the Fl_Spinner widget.

See also

void wrap(int)

Since

1.4.0

12.132.3.10 wrap() [2/2]

void Fl_Spinner::wrap (int set) [inline]

Sets whether the spinner wraps around at upper and lower bounds. If wrap mode is on the spinner value is set to the minimum() or maximum() if the value exceeds the upper or lower bounds, resp., if it was changed by one of the buttons or the FL_Up or FL_Down keys. The spinner stops at the upper and lower bounds if wrap mode is off. The default wrap mode is on for backwards compatibility with FLTK 1.3.x and older versions.
Note

Wrap mode does not apply to the input field if the input value is edited directly as a number. The input value is always clipped to the allowed range as if wrap mode was off when the input field is left (i.e. loses focus).

See also

minimum(), maximum()

Parameters

```
in  set  non-zero sets wrap mode, zero resets wrap mode
```

Since

1.4.0

The documentation for this class was generated from the following files:

- Fl_Spinner.H
- Fl_Spinner.cxx

### 12.133 Fl_Spinner::Fl_Spinner_Input Class Reference

Inheritance diagram for Fl_Spinner::Fl_Spinner_Input:

```
Fl_Widget
  |
  |
Fl_Input_
  |
  |
Fl_Input
  |
  |
Fl_Spinner::Fl_Spinner_INPUT
```

**Public Member Functions**

- **Fl_Spinner_Input** (int X, int Y, int W, int H)
- **int handle** (int event) FL_OVERRIDE
  
  Handles events of Fl_Spinner’s embedded input widget.

**Public Member Functions inherited from Fl_Input**

- **Fl_Input** (int, int, int, int, const char ∗=0)
  
  Creates a new Fl_Input widget using the given position, size, and label string.

**Public Member Functions inherited from Fl_Input_**

- **int append** (const char ∗t, int l=0, char keep_selection=0)
  
  Append text at the end.
- **bool can_redo** () const
  
  Check if there is a redo action available.
- **bool can_undo** () const
  
  Check if the last operation can be undone.
• int **copy** (int clipboard)
  
  *Put the current selection into the clipboard.*

• int **copy_cuts** ()
  
  *Copies the yank buffer to the clipboard.*

• **Fl_Color cursor_color** () const
  
  *Gets the color of the cursor.*

• void **cursor_color** (Fl_Color n)
  
  *Sets the color of the cursor.*

• int **cut** ()
  
  *Deletes the current selection.*

• int **cut** (int a, int b)
  
  *Deletes all characters between index a and b.*

• int **cut** (int n)
  
  *Deletes the next n bytes rounded to characters before or after the cursor.*

• double **dvalue** () const
  
  *Returns the widget text interpreted as a floating point number.*

• Fl_Input_ (int, int, int, const char ∗=0)
  
  *Creates a new Fl_Input_ widget.*

• unsigned int **index** (int i) const
  
  *Returns the character at index i.*

• int **input_type** () const
  
  *Gets the input field type.*

• void **input_type** (int t)
  
  *Sets the input field type.*

• int **insert** (const char ∗t, int l=0)
  
  *Inserts text at the cursor position.*

• int **insert_position** () const
  
  *Gets the position of the text cursor.*

• int **insert_position** (int p)
  
  *Sets the cursor position and mark.*

• int **insert_position** (int p, int m)
  
  *Sets the index for the cursor and mark.*

• int **ivalue** () const
  
  *Returns the widget text interpreted as a signed integer.*

• int **mark** () const
  
  *Gets the current selection mark.*

• int **mark** (int m)
  
  *Sets the current selection mark.*

• int **maximum_size** () const
  
  *Gets the maximum length of the input field in characters.*

• void **maximum_size** (int m)
  
  *Sets the maximum length of the input field in characters.*

• int **position** () const
  
  *Gets the read-only state of the input field.*

• void **readonly** (int b)
  
  *Sets the read-only state of the input field.*

• int **redo** ()
  
  *Redo previous undo operation.*


- **int replace**(int b, int e, const char *text, int ilen=0)
  
  Deletes text from b to e and inserts the new string text.

- **void resize**(int, int, int, int) FL_OVERRIDE
  
  Changes the size of the widget.

- **int shortcut**( ) const
  
  Return the shortcut key associated with this widget.

- **void shortcut**(int s)
  
  Sets the shortcut key associated with this widget.

- **int size**( ) const
  
  Returns the number of bytes in value().

- **void size**(int W, int H)
  
  Sets the width and height of this widget.

- **int static_value**(const char *)
  
  Changes the widget text.

- **int static_value**(const char *, int)
  
  Changes the widget text.

- **int tab_nav**( ) const
  
  Gets whether the Tab key causes focus navigation in multiline input fields or not.

- **void tab_nav**(int val)
  
  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.

- **Fl_Color textcolor**( ) const
  
  Gets the color of the text in the input field.

- **void textcolor**(Fl_Color n)
  
  Sets the color of the text in the input field.

- **Fl_Font textfont**( ) const
  
  Gets the font of the text in the input field.

- **void textfont**(Fl_Font s)
  
  Sets the font of the text in the input field.

- **Fl_Fontsize textsize**( ) const
  
  Gets the size of the text in the input field.

- **void textsize**(Fl_Fontsize s)
  
  Sets the size of the text in the input field.

- **int undo**( )
  
  Undoes previous changes to the text buffer.

- **const char * value**( ) const
  
  Returns the text displayed in the widget.

- **int value**(const char *)
  
  Changes the widget text.

- **int value**(const char *, int)
  
  Changes the widget text.

- **int value**(double value)
  
  Changes the widget text to a floating point number ("%g").

- **int value**(int value)
  
  Changes the widget text to a signed integer number.

- **int wrap**( ) const
  
  Gets the word wrapping state of the input field.

- **void wrap**(int b)
  
  Sets the word wrapping state of the input field.

- **~Fl_Input**( )
  
  Destroys the widget.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ as_gl_window () const
- virtual Fl_Group ∗ as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ as_group () const
- virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ as_window () const
- void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
Checks if the widget value changed since the last callback.

- **void clear_active()**
  Marks the widget as inactive without sending events or changing focus.

- **void clear_changed()**
  Marks the value of the widget as unchanged.

- **void clear_damage(uchar c=0)**
  Clears or sets the damage flags.

- **void clear_output()**
  Sets a widget to accept input.

- **void clear_visible()**
  Hides the widget.

- **void clear_visible_focus()**
  Disables keyboard focus navigation with this widget.

- **Fl_Color color() const**
  Gets the background color of the widget.

- **void color(Fl_Color bg)**
  Sets the background color of the widget.

- **void color(Fl_Color bg, Fl_Color sel)**
  Sets the background and selection color of the widget.

- **Fl_Color color2() const**
  For back compatibility only.

- **void color2(unsigned a)**
  For back compatibility only.

- **int contains(const Fl_Widget *w) const**
  Checks if w is a child of this widget.

- **void copy_label(const char *new_label)**
  Sets the current label.

- **void copy_tooltip(const char *text)**
  Sets the current tooltip text.

- **uchar damage() const**
  Returns non-zero if draw() needs to be called.

- **void damage(uchar c)**
  Sets the damage bits for the widget.

- **void damage(uchar c, int x, int y, int w, int h)**
  Sets the damage bits for an area inside the widget.

- **int damage_resize(int, int, int, int)**
  Internal use only.

- **void deactivate()**
  Deactivates the widget.

- **Fl_Image *deimage()**
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image *deimage() const**
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage(Fl_Image &img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage(Fl_Image *img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound() const**
  Returns whether the inactive image is managed by the widget.

- **void do_callback(Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image *image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image *image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char *label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
Sets whether this widget needs a keyboard.
• unsigned int `output () const`
  Returns if a widget is used for output only.
• `Fl_Group * parent () const`
  Returns a pointer to the parent widget.
• void `parent (Fl_Group *p)`
  Internal use only - "for hacks only".
• void `position (int X, int Y)`
  Repositions the window or widget.
• void `re redraw ()`
  Schedules the drawing of the widget.
• void `re redraw_label ()`
  Schedules the drawing of the label.
• `Fl_Color selection_color () const`
  Gets the selection color.
• void `selection_color (Fl_Color a)`
  Sets the selection color.
• void `set_active ()`
  Marks the widget as active without sending events or changing focus.
• void `set_changed ()`
  Marks the value of the widget as changed.
• void `set_output ()`
  Sets a widget to output only.
• void `set_visible ()`
  Makes the widget visible.
• void `set_visible_focus ()`
  Enables keyboard focus navigation with this widget.
• int `shortcut_label () const`
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void `shortcut_label (int value)`
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void `show ()`
  Makes a widget visible.
• void `size (int W, int H)`
  Changes the size of the widget.
• int `take_focus ()`
  Gives the widget the keyboard focus.
• unsigned int `takeevents () const`
  Returns if the widget is able to take events.
• int `test_shortcut ()`
  Returns true if the widget's label contains the entered '&x' shortcut.
• const char * `tooltip () const`
  Gets the current tooltip text.
• void `tooltip (const char *text)`
  Sets the current tooltip text.
• `Fl_Window * top_window () const`
  Returns a pointer to the top-level window for the widget.
• `Fl_Window * top_window_offset (int &xoff, int &yoff) const`
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar `type () const`
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
• static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
• static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Static Public Attributes inherited from Fl_Input

• static const char * copy_menu_text = "Copy"
  [this text may be customized at run-time]
• static const char * cut_menu_text = "Cut"
  [this text may be customized at run-time]
• static const char * paste_menu_text = "Paste"
  [this text may be customized at run-time]
Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

Protected Member Functions inherited from Fl_Input

- void draw () FL_OVERRIDE
  Draws the widget.
- int handle_key ()
  Handles a keystroke.
- int handle_rmb ()
  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

- int apply_undo ()
  Apply the current undo/redo operation.
- void drawtext (int, int, int, int)
  Draws the text in the passed bounding box.
- void drawtext (int, int, int, int, bool draw_active)
  Draws the text in the passed bounding box.
- void handle_mouse (int, int, int, int, int keepmark=0)
  Handles mouse clicks and mouse moves.
- int handletext (int e, int, int, int)
  Handles all kinds of text field related events.
- int line_end (int i) const
  Finds the end of a line.
- int line_start (int i) const
  Finds the start of a line.
- int linesPerPage ()
  Returns the number of lines per page.
- void maybe_do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
- int up_down_position (int, int keepmark=0)
  Moves the cursor to the column given by up_down_pos.
- int word_end (int i) const
  Finds the end of a word.
- int word_start (int i) const
  Finds the start of a word.
- int xscroll () const
- int yscroll () const
- void yscroll (int yOffset)
Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)
  Clears a flag in the flags mask.

- void **draw_backdrop** () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void **draw_box** () const
  Draws the widget box according its box style.

- void **draw_box** (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- void **draw_focus** () const
  Draws a focus rectangle around the widget.

- void **draw_focus** (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

- void **draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

- void **draw_label** () const
  Draws the widget's label at the defined label position.

- void **draw_label** (int, int, int, int) const
  Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.

- unsigned int **flags** () const
  Gets the widget flags mask.

- void **h** (int v)
  Internal use only.

- void **set_flag** (unsigned int c)
  Sets a flag in the flags mask.

- void **w** (int v)
  Internal use only.

- void **x** (int v)
  Internal use only.

- void **y** (int v)
  Internal use only.

### 12.133.1 Member Function Documentation

#### 12.133.1.1 handle()

```cpp
int Fl_Spinner::Fl_Spinner_Input::handle (int event) [virtual]
```

Handles events of Fl_Spinner's embedded input widget.
Works like Fl_Input::handle() but ignores FL_Up and FL_Down keys so they can be handled by the parent widget (Fl_Spinner).
Reimplemented from Fl_Input.
The documentation for this class was generated from the following files:

- Fl_Spinner.H
- Fl_Spinner.cxx
12.134  Fl_Surface_Device Class Reference

A drawing surface that’s susceptible to receive graphical output.
#include <Fl_Device.H>

Inheritance diagram for Fl_Surface_Device:

```
Fl_Surface_Device
   Fl_Display_Device Fl_Widget_Surface
   Fl_Copy_Surface Fl_EPS_File_Surface Fl_Image_Surface Fl_Paged_Device Fl_SVG_File_Surface
   Fl_PostScript_File_Device Fl_Printer
```

Public Member Functions

- `Fl_Graphics_Driver * driver()`
  Returns the graphics driver of this drawing surface.
- `virtual bool is_current()`
  Is this surface the current drawing surface?
- `virtual void set_current(void)`
  Make this surface the current drawing surface.
- `virtual ~Fl_Surface_Device()`
  The destructor.

Static Public Member Functions

- `static Fl_Surface_Device * pop_current()`
  Removes the top element from the current drawing surface stack, and makes the new top element current.
- `static void push_current(Fl_Surface_Device *new_current)`
  Pushes new_current on top of the stack of current drawing surfaces, and makes it current.
- `static Fl_Surface_Device * surface()`
  The current drawing surface.

Protected Member Functions

- `void driver(Fl_Graphics_Driver *graphics_driver)`
  Sets the graphics driver of this drawing surface.
- `virtual void end_current()`
  FLTK calls this each time a surface ceases to be the current drawing surface.
- `Fl_Surface_Device (Fl_Graphics_Driver *graphics_driver)`
  Constructor that sets the graphics driver to use for the created surface.

12.134.1  Detailed Description

A drawing surface that’s susceptible to receive graphical output.

Any FLTK application has at any time a current drawing surface to which all drawing requests are directed. The
current surface is given by Fl_Surface_Device::surface(). When main() begins running, the current drawing surface
has been set to the computer’s display, an instance of the Fl_Display_Device class.

A drawing surface other than the computer’s display, is typically used as follows:

1. Create `surface`, an object from a particular Fl_Surface_Device derived class (e.g., Fl_Copy_Surface,
Fl_Printer).
2. Call `Fl_Surface_Device::push_current(surface);` to redirect all graphics requests to
   `surface` which becomes the new current drawing surface (not necessary with classes Fl_Printer /
   Fl_PostScript_File_Device because it is done by Fl_Paged_Device::begin_page()).

Generated by Doxygen
3. At this point all of the Drawing functions (e.g., \texttt{fl_rect()}) or the Color & Font functions or Drawing Images functions (e.g., \texttt{fl_draw_image()}, \texttt{Fl_Image::draw()}) operate on the new current drawing surface. Drawing surfaces from \texttt{Fl_Widget_Surface} derived classes allow additional ways to draw to them (e.g., \texttt{Fl_Printer::print_widget()}, \texttt{Fl_Image_Surface::draw()}).

4. After all drawing requests have been performed, redirect graphics requests back to their previous destination with \texttt{Fl_Surface_Device::pop_current();} (not necessary with classes \texttt{Fl_Printer} / \texttt{Fl_PostScript_File_Device}).

5. Delete surface.

For back-compatibility, it is also possible to use the \texttt{Fl_Surface_Device::set_current()} member function to change the current drawing surface, once to the new surface, once to the previous one.

Class \texttt{Fl_Surface_Device} can also be derived to define new kinds of graphical output usable with FLTK drawing functions. An example would be to draw to a PDF file. This would require to create a new class, say PDF\_File Surface, derived from class \texttt{Fl_Surface_Device}, and another new class, say PDF\_Graphics\_Driver, derived from class Fl\_Graphics\_Driver. Class PDF\_Graphics\_Driver should implement all virtual methods of the Fl\_Graphics\_Driver class to support all FLTK drawing functions and have them draw into PDF files. Alternatively, class PDF\_Graphics\_Driver could implement only some virtual methods, and only part of the FLTK drawing API would be usable when drawing to PDF files.

### 12.134.2 Member Function Documentation

#### 12.134.2.1 \texttt{end_current()} 

virtual void \texttt{Fl_Surface_Device::end_current()} [inline], [protected], [virtual]

FLTK calls this each time a surface ceases to be the current drawing surface.

This member function is mostly of interest to developers of new Fl\_Surface\_Device derived classes. It allows to perform surface-specific operations necessary when this surface ceases to be current. Each implementation should end with a call to \texttt{Fl_Surface_Device::end_current()}.

Reimplemented in \texttt{Fl_PostScript_File_Device}.

#### 12.134.2.2 \texttt{is_current()} 

bool \texttt{Fl_Surface_Device::is_current()} [virtual]

Is this surface the current drawing surface?

Reimplemented in \texttt{Fl_Copy_Surface, Fl_Image_Surface, and Fl_Printer}.

#### 12.134.2.3 \texttt{pop_current()} 

\texttt{Fl_Surface_Device \* Fl_Surface_Device::pop_current()} [static]

Removes the top element from the current drawing surface stack, and makes the new top element current.

Returns

A pointer to the new current drawing surface.

See also

\texttt{Fl_Surface_Device::push_current(Fl_Surface_Device \*)}

Version

1.4.0

#### 12.134.2.4 \texttt{push_current()} 

void \texttt{Fl_Surface_Device::push_current} ( 

\texttt{Fl_Surface_Device \* new_current} ) [static]

Pushes \texttt{new_current} on top of the stack of current drawing surfaces, and makes it current.

\texttt{new_current} will receive all future graphics requests.

Any call to \texttt{push_current()} must be matched by a subsequent call to \texttt{Fl_Surface_Device::pop_current()}. The max height of this stack is 16.
12.134.2.5 set_current()

```cpp
void Fl_Surface_Device::set_current ( void ) [virtual]
```

Make this surface the current drawing surface. This surface will receive all future graphics requests. Starting from FLTK 1.4.0, the preferred API to change the current drawing surface is `Fl_Surface_Device::push_current( ) / Fl_Surface_Device::pop_current()`.

**Note**

It's recommended to use this function only as follows:

- The current drawing surface is the display;
- make current another surface, e.g., an `Fl_Printer` or an `Fl_Image_Surface` object, calling `set_current()` on this object;
- draw to that surface;
- make the display current again with `Fl_Display_Device::display_device()->set_current()`. Don't do any other call to `set_current()` before this one.

Other scenarios of drawing surface changes should be performed via `Fl_Surface_Device::push_current( ) / Fl_Surface_Device::pop_current()`.

Reimplemented in `Fl_Copy_Surface`, `Fl_Image_Surface`, `Fl_PostScript_File_Device`, and `Fl_Printer`.

12.134.2.6 surface()

```cpp
static Fl_Surface_Device * Fl_Surface_Device::surface ( ) [inline], [static]
```

The current drawing surface. In other words, the `Fl_Surface_Device` object that currently receives all graphics requests.

**Note**

It's possible to transiently remove the GUI scaling factor in place in the current drawing surface with `fl_override_scale()`.

The documentation for this class was generated from the following files:

- `Fl_Device.H`
- `Fl_Device.cxx`

# 12.135 Fl_SVG_File_Surface Class Reference

A drawing surface producing a Scalable Vector Graphics (SVG) file.

```cpp
#include <Fl_SVG_File_Surface.H>
```

Inheritance diagram for `Fl_SVG_File_Surface`:

```
Fl_Surface_Device
   |
   V
Fl_Widget_Surface
   |
   V
Fl_SVG_File_Surface
```
Public Member Functions

- `int close ()`
  Closes the FILE pointer where SVG data is output.

- `FILE * file ()`
  Returns the underlying FILE pointer.

- `Fl_SVG_File_Surface (int width, int height, FILE *svg, int(*closef)(FILE *))=NULL`
  Constructor of the SVG drawing surface.

- `void origin (int *x, int *y) FL_OVERRIDE`
  Computes the coordinates of the current origin of graphics functions.

- `void origin (int x, int y) FL_OVERRIDE`
  Sets the position of the origin of graphics in the drawable part of the drawing surface.

- `int printable_rect (int *w, int *h) FL_OVERRIDE`
  Computes the width and height of the drawable area of the drawing surface.

- `void translate (int x, int y) FL_OVERRIDE`
  Translates the current graphics origin accounting for the current rotation.

- `void untranslate () FL_OVERRIDE`
  Undoes the effect of a previous translate() call.

- `~Fl_SVG_File_Surface ()`
  Destructor.

Public Member Functions inherited from Fl_Widget_Surface

- `void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)`
  Draws the widget on the drawing surface.

- `void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)`
  Draws a window with its title bar and frame if any.

- `void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)`
  Draws a rectangular part of an on-screen window.

Public Member Functions inherited from Fl_Surface_Device

- `Fl_Graphics_Driver * driver ()`
  Returns the graphics driver of this drawing surface.

- `virtual bool is_current ()`
  Is this surface the current drawing surface?

- `virtual void set_current (void)`
  Make this surface the current drawing surface.

- `virtual ~Fl_Surface_Device ()`
  The destructor.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device

- `static Fl_Surface_Device * pop_current ()`
  Removes the top element from the current drawing surface stack, and makes the new top element current.

- `static void push_current (Fl_Surface_Device *new_current)`
  Pushes new_current on top of the stack of current drawing surfaces, and makes it current.

- `static Fl_Surface_Device * surface ()`
  The current drawing surface.
Protected Member Functions inherited from Fl_Widget_Surface

• **Fl_Widget_Surface** (Fl_Graphics_Driver *d)
  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

• **void driver** (Fl_Graphics_Driver *graphics_driver)
  Sets the graphics driver of this drawing surface.

• virtual **void end_current ()**
  FLTK calls this each time a surface ceases to be the current drawing surface.

• **Fl_Surface_Device** (Fl_Graphics_Driver *graphics_driver)
  Constructor that sets the graphics driver to use for the created surface.

Protected Attributes inherited from Fl_Widget_Surface

• **int x_offset**
  horizontal offset to the origin of graphics coordinates

• **int y_offset**
  vertical offset to the origin of graphics coordinates

12.135.1 Detailed Description

A drawing surface producing a Scalable Vector Graphics (SVG) file.
This drawing surface allows to store any FLTK graphics in vectorial form in a “Scalable Vector Graphics” file.
Usage example:

```cpp
Fl_Window *win = ... // Window to draw to a .svg file
int ww = win->decorated_w();
int wh = win->decorated_h();
FILE *svg = fl_fopen("/path/to/mywindow.svg", "w");
if (svg) {
  Fl_SVG_File_Surface *surface = new Fl_SVG_File_Surface(ww, wh, svg);
  Fl_Surface_Device::push_current(surface);
  fl_color(FL_WHITE);
  fl_rectf(0, 0, ww, wh);
  surface->draw_decorated_window(win);
  Fl_Surface_Device::pop_current();
  delete surface; // the .svg file is not complete until the destructor was run
  fclose(svg);
}
```

Note
FLTK uses the PNG and JPEG libraries to encode images to the SVG format. For this reason, class Fl_SVG_File_Surface is placed in the fltk_images library. If JPEG is not available at application build time, PNG is enough (but produces a quite larger output). If PNG isn't available either, images don't appear in the SVG output.

12.135.2 Constructor & Destructor Documentation

12.135.2.1 **Fl_SVG_File_Surface()**

Fl_SVG_File_Surface::Fl_SVG_File_Surface (  
  int width,  
  int height,  
  FILE * svg,  
  int(*)(FILE *) closef = NULL  )

Constructor of the SVG drawing surface.

Parameters

| width, height | Width and height of the graphics area in FLTK drawing units |

Generated by Doxygen
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>svg</code></td>
<td>A writable FILE pointer where the SVG data are to be sent. The resulting SVG data are not complete until after destruction of the Fl_SVG_File_Surface object or after calling close().</td>
</tr>
<tr>
<td><code>closef</code></td>
<td>If not NULL, the destructor and close() will call closef(svg) after all SVG data has been sent. If NULL, fclose(svg) is called instead. This allows to close the FILE pointer by, e.g., pclose, or, using a function such as &quot;int keep_open(FILE *){return 0;}&quot;, to keep it open after completion of all output to svg. Function closef should return non zero to indicate an error.</td>
</tr>
</tbody>
</table>

#### 12.135.2.2 ~Fl_SVG_File_Surface()

Fl_SVG_File_Surface::~Fl_SVG_File_Surface ( )

Destructor.
The underlying FILE pointer is processed as by close().

#### 12.135.3 Member Function Documentation

##### 12.135.3.1 close()

```c
int Fl_SVG_File_Surface::close ( )
```

Closes the FILE pointer where SVG data is output.
The underlying FILE is closed by function fclose() unless another function was set at object’s construction time. The only operation possible after this on the Fl_SVG_File_Surface object is its destruction.

**Returns**
The value returned by the closing function call.

##### 12.135.3.2 origin() [1/2]

```c
void Fl_SVG_File_Surface::origin ( int * x, int * y ) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out x,y</code></td>
<td>If non-null, +x and +y are set to the horizontal and vertical coordinates of the graphics origin.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Widget_Surface.

##### 12.135.3.3 origin() [2/2]

```c
void Fl_SVG_File_Surface::origin ( int x, int y ) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface. Arguments should be expressed relatively to the result of a previous printable_rect() call. That is, printable_rect(&w, &h); origin(w/2, 0); sets the graphics origin at the top center of the drawable area. Successive origin() calls don’t combine their effects. Origin() calls are not affected by rotate() calls (for classes derived from Fl_Paged_Device).

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in x,y</code></td>
<td>Horizontal and vertical positions in the drawing surface of the desired origin of graphics.</td>
</tr>
</tbody>
</table>
Reimplemented from Fl_Widget_Surface.

12.135.3.4 printable_rect()

```cpp
int Fl_SVG_File_Surface::printable_rect (int * w, int * h) [virtual]
```
Computes the width and height of the drawable area of the drawing surface. Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

12.135.3.5 translate()

```cpp
void Fl_SVG_File_Surface::translate (int x, int y) [virtual]
```
Translates the current graphics origin accounting for the current rotation. Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.

Reimplemented from Fl_Widget_Surface.

12.135.3.6 untranslate()

```cpp
void Fl_SVG_File_Surface::untranslate (void ) [virtual]
```
Undoes the effect of a previous translate() call.

Reimplemented from Fl_Widget_Surface.

The documentation for this class was generated from the following file:

- Fl_SVG_File_Surface.H

12.136 Fl_SVG_Image Class Reference

The Fl_SVG_Image class supports loading, caching and drawing of scalable vector graphics (SVG) images.

```
#include <Fl_SVG_Image.H>
```

Inheritance diagram for Fl_SVG_Image:

```
  Fl_SVG_Image
    |      |
    Fl_RGB_Image
      |      |
      Fl_SVG_Image
```

Public Member Functions

- `Fl_SVG_Image * as_svg_image () FL_OVERRIDE`
  Returns whether an image is an Fl_SVG_Image or not.
- `void color_average (Fl_Color c, float i) FL_OVERRIDE`
  The color_average() method averages the colors in the image with the provided FLTK color value.
- `Fl_Image * copy () const`
• **FL_Image** » **copy** (int W, int H) const **FL_OVERRIDE**
  Creates a resized copy of the image.
• void **desaturate** () **FL_OVERRIDE**
  The **desaturate**() method converts an image to grayscale.
• void **draw** (int X, int Y)
• void **draw** (int X, int Y, int W, int H, int cx=0, int cy=0) **FL_OVERRIDE**
  Draws the image to the current drawing surface with a bounding box.
• **FL_SVG_Image** (const char ∗filename)
  Load an SVG image from a file.
• **FL_SVG_Image** (const char ∗sharedname, const char ∗svg_data)
  Load an SVG image from memory.
• **FL_SVG_Image** (const char ∗sharedname, const unsigned char ∗svg_data, size_t length)
  Load an SVG image from memory.
• void **normalize** () **FL_OVERRIDE**
  Makes sure the object is fully initialized.
• void **resize** (int width, int height)
  Have the svg data (re-)rasterized using the given width and height values.
• virtual ~**FL_SVG_Image** ()
  The destructor frees all memory and server resources that are used by the SVG image.

**Public Member Functions inherited from FL_RGB_Image**

• int **cache_h** ()
• int **cache_w** ()
• **FL_Image** » **copy** () const
• void **draw** (int X, int Y)
• **FL_RGB_Image** (const Fl_Pixmap ∗pxm, Fl_Color bg=FL_GRAY)
  The constructor creates a new RGBA image from the specified Fl_Pixmap.
• **FL_RGB_Image** (const uchar ∗bits, int bits_length, int W, int H, int D, int LD)
  The constructor creates a new image from the specified data.
• **FL_RGB_Image** (const uchar ∗bits, int W, int H, int D=3, int LD=0)
  The constructor creates a new image from the specified data.
• void **label** (Fl_Menu_Item ∗m) **FL_OVERRIDE**
  This method is an obsolete way to set the image attribute of a menu item.
• void **label** (Fl_Widget ∗w) **FL_OVERRIDE**
  This method is an obsolete way to set the image attribute of a widget or menu item.
• void **uncache** () **FL_OVERRIDE**
  If the image has been cached for display, delete the cache data.
• ~**FL_RGB_Image** () **FL_OVERRIDE**
  The destructor frees all memory and server resources that are used by the image.

**Public Member Functions inherited from FL_Image**

• virtual class Fl_Shared_Image ∗ **as_shared_image** ()
  Returns whether an image is an Fl_Shared_Image or not.
• **FL_Image** » **copy** () const
  Creates a copy of the image in the same size.
• int **count** () const
  Returns the number of data values associated with the image.
• int **d** () const
  Returns the image depth.
• const char ∗const + **data** () const
Returns a pointer to the current image data array.

- **int data_h ()** const
  Returns the height of the image data.

- **int data_w ()** const
  Returns the width of the image data.

- **void draw (int X, int Y)**
  Draws the image to the current drawing surface.

- **int fail ()** const
  Returns a value that is not 0 if there is currently no image available.

- **Fl_Image (int W, int H, int D)**
  The constructor creates an empty image with the specified width, height, and depth.

- **int h ()** const
  Returns the current image drawing height in FLTK units.

- **void inactive ()**
  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

- **int ld ()** const
  Returns the current line data size in bytes.

- **virtual void release ()**
  Releases an Fl_Image - the same as 'delete this'.

- **virtual void scale (int width, int height, int proportional=1, int can_expand=0)**
  Sets the drawing size of the image.

- **int w ()** const
  Returns the current image drawing width in FLTK units.

- **virtual ~Fl_Image ()**
  The destructor is a virtual method that frees all memory used by the image.

**Public Attributes**

- **bool proportional**
  Set this to false to allow image re-scaling that alters the image aspect ratio.

**Public Attributes inherited from Fl_RGB_Image**

- **int alloc_array**
  If non-zero, the object's data array is delete[]d when deleting the object.

- **const uchar * array**
  Points to the start of the object's data array.

**Additional Inherited Members**

**Static Public Member Functions inherited from Fl_RGB_Image**

- **static size_t max_size ()**
  Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

- **static void max_size (size_t size)**
  Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
12.136 Fl_SVG_Image Class Reference

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype define_FL_IMAGE_LABEL()
- static Fl_RGB_Scaling RGB_scaling()
  
  Returns the currently used RGB image scaling method.
- static void RGB_scaling(Fl_RGB_Scaling)
  
  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling scaling_algorithm()
  
  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm(Fl_RGB_Scaling algorithm)
  
  Sets what algorithm is used when resizing a source image to draw it.

Static Public Attributes inherited from Fl_Image

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_MEMORY_ACCESS = -4
- static const int ERR_NO_IMAGE = -1
- static bool register_images_done = false
  
  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_Image

- void d(int D)
  
  Sets the current image depth.
- void data(const char *const *p, int c)
  
  Sets the current data pointer and count of pointers in the array.
- void draw_empty(int X, int Y)
  
  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled(int X, int Y, int W, int H)
  
  Draw the image to the current drawing surface rescaled to a given width and height.
- void h(int H)
  
  Sets the height of the image data.
- void ld(int LD)
  
  Sets the current line data size in bytes.
- void w(int W)
  
  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void labeltype(const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void measure(const Fl_Label *lo, int &lw, int &lh)

12.136.1 Detailed Description

The Fl_SVG_Image class supports loading, caching and drawing of scalable vector graphics (SVG) images. The FLTK library performs parsing and rasterization of SVG data using a modified version of the nanosvg software ([https://github.com/memononen/nanosvg](https://github.com/memononen/nanosvg)). The software modification allows the option to change the image ratio while performing rasterization.

Use Fl_Image::fail() to check if the Fl_SVG_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, and ERR_FORMAT if the SVG format could not be decoded. If the image has loaded correctly, w(), h(), and d() should return values greater than zero.

Rasterization is not done until the image is first drawn or resize() or normalize() is called. Therefore, array is NULL until then. The delayed rasterization ensures an Fl_SVG_Image is always rasterized to the exact screen resolution at which it is drawn.

The Fl_SVG_Image class draws images computed by nanosvg with the following known limitations.
The FLTK library can optionally be built without SVG support; in that case, class Fl_SVG_Image is unavailable.

Example of displaying a hard-coded svg file:

```c
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_SVG_Image.H>

// A black rotated rectangle
const char *svg_data = "<svg viewBox="0 0 200 200" version = "1.1" xmlns="http://www.w3.org/2000/svg">
  
  <rect x="25" y="50" width="150" height="100" fill="black" transform="rotate(45 100 100)" />
</svg>";

int main(int argc, char **argv) {
    Fl_SVG_Image *svg = new Fl_SVG_Image(0, svg_data); // create SVG object
    Fl_Window *win = new Fl_Window(720, 486, "svg test");
    Fl_Box *box = new Fl_Box(0, 0, win->w(), win->h());
    box->image(svg); // assign svg object to Fl_Box
    win->end();
    win->show(argc,argv);
    return(Fl::run());
}
```

Example of displaying an svg image from a file:

```c
#include <errno.h> // errno
#include <string.h> // strerror
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_SVG_Image.H>
#include <FL/fl_message.H>

int main(int argc, char **argv) {
    Fl_Window *win = new Fl_Window(720, 486, "svg test");
    Fl_Box *box = new Fl_Box(0, 0, win->w(), win->h());
    const char *svgpath = "/var/tmp/simple.svg";
    Fl_SVG_Image *svg = new Fl_SVG_Image(svgpath); // load SVG object from disk
    switch (svg->fail()) { // File couldn't load? show path + os error to user
        case Fl_Image::ERR_FILE_ACCESS:
            // File couldn't load? show path + os error to user
            fl_alert("%s: %s", svgpath, strerror(errno));
            return 1;
        case Fl_Image::ERR_FORMAT:
            // Parsing error
            fl_alert("%s: couldn't decode image", svgpath);
            return 1;
    }
    box->image(svg); // assign svg object to box
    win->end();
    win->show(argc,argv);
    return(Fl::run());
}
```

Example of fitting an svg image to a resizable Fl_Box:

```c
#include <FL/Fl_Window.H>
#include <FL/Fl_SVG_Image.H>
#include <FL/Fl_Box.H>

class resizable_box : public Fl_Box {
public:
    resizable_box(int w, int h) : Fl_Box(0, 0, w, h, NULL) {}  
    virtual void resize(int x, int y, int w, int h) { 
        image()->scale(w, h, 1, 1); // p3 = proportional, p4 = can expand
        Fl_Box::resize(x, y, w, h);
    }
};

int main(int argc, char **argv) {
    Fl_Window *win = new Fl_Window(130, 130);
    resizable_box *box = new resizable_box(win->w(), win->h());
    Fl_SVG_Image *svg = new Fl_SVG_Image("/path/to/image.svg");
    box->image(svg);
    svg->scale(box->w(), box->h());
    win->end();
    win->resizable(win);
    win->show(argc, argv);
    return Fl::run();
}
```
12.136.2 Constructor & Destructor Documentation

12.136.2.1 Fl_SVG_Image() [1/3]

Fl_SVG_Image::Fl_SVG_Image (const char ∗filename)

Load an SVG image from a file.
This constructor loads the SVG image from a .svg or .svgz file. The reader recognizes if the data is compressed, and decompresses it if zlib is available (HAVE_LIBZ).

Parameters

- **filename** the filename for a .svg or .svgz file

12.136.2.2 Fl_SVG_Image() [2/3]

Fl_SVG_Image::Fl_SVG_Image (const char ∗sharedname, const char ∗svg_data)

Load an SVG image from memory.
This constructor loads the SVG image from a block of memory. This version is commonly used for uncompressed text data, but the reader recognizes if the data is compressed, and decompresses it if zlib is available (HAVE_LIBZ).

Parameters

- **sharedname** if not NULL, a shared image will be generated with this name
- **svg_data** a pointer to the memory location of the SVG image data

Note

In-memory SVG data is parsed by the object constructor and is no longer needed after construction.

12.136.2.3 Fl_SVG_Image() [3/3]

Fl_SVG_Image::Fl_SVG_Image (const char ∗name, const unsigned char ∗svg_data, size_t length)

Load an SVG image from memory.
This constructor loads the SVG image from a block of memory. This version is commonly used for compressed binary data, but the reader recognizes if the data is uncompressed, and reads it as a text block.

Parameters

- **name** if not NULL, a shared image will be generated with this name
- **svg_data** a pointer to the memory location of the SVG image data
- **length** of svg_data or 0 if the length is unknown. This will protect memory outside of the svg_data array from illegal read operations for compressed SVG data.
Note

In-memory SVG data is parsed by the object constructor and is no longer needed after construction.

12.136.3 Member Function Documentation

12.136.3.1 as_svg_image()

Fl_SVG_Image ∗ Fl_SVG_Image::as_svg_image ( ) [inline], [virtual]

Returns whether an image is an Fl_SVG_Image or not.
This virtual method returns a pointer to the Fl_SVG_Image if this object is an instance of Fl_SVG_Image or NULL if not.
Reimplemented from Fl_RGB_Image.

12.136.3.2 color_average()

void Fl_SVG_Image::color_average ( Fl_Color c,
   float i ) [virtual]

The color_average() method averages the colors in the image with the provided FLTK color value.
The first argument specifies the FLTK color to be used.
The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results
in no color blend, and a value of 0.0 results in a constant image of the specified color.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original
image data in memory.
Reimplemented from Fl_RGB_Image.

12.136.3.3 copy()

Fl_Image ∗ Fl_SVG_Image::copy ( int W,
   int H ) const [virtual]

Creates a resized copy of the image.
The new image should be released when you are done with it.
Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image)
instead of operator delete for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's.
The new image data will be converted to the requested size. RGB images are resized using the algorithm set by
Fl_Image::RGB_scaling().
For the new image the following equations are true:

\[ w() == \text{data}_w() == W \]
\[ h() == \text{data}_h() == H \]

Parameters

| in | W/H | Requested width and height of the new image |

Note

The returned image can be safely cast to the same image type as that of the source image provided this type
is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent
class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.
Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your
overridden methods of 'Fl_Image::copy(int, int) const' and 'Fl_Image::copy() const' must also be 'const' for
inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not
been 'const'.
Reimplemented from Fl_RGB_Image.
12.136.3.4 desaturate()

```cpp
void Fl_SVG_Image::desaturate ( ) [virtual]
```

The `desaturate()` method converts an image to grayscale. If the image contains an alpha channel (depth = 4), the alpha channel is preserved. An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.

Reimplemented from `Fl_RGB_Image`.

12.136.3.5 draw()

```cpp
void Fl_SVG_Image::draw ( int X, int Y, int W, int H, int cx = 0, int cy = 0 ) [virtual]
```

Draws the image to the current drawing surface with a bounding box. Arguments X,Y,W,H specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the cx and cy arguments.

In other words: `fl_push_clip(X,Y,W,H)` is applied, the image is drawn with its upper-left corner at X-cx,Y-cy and its own width and height, `fl_pop_clip()` is applied.

Reimplemented from `Fl_RGB_Image`.

12.136.3.6 normalize()

```cpp
void Fl_SVG_Image::normalize ( ) [virtual]
```

Makes sure the object is fully initialized. This function rasterizes the SVG image, and consequently initializes its array member, if that was not done before.

Reimplemented from `Fl_RGB_Image`.

12.136.3.7 resize()

```cpp
void Fl_SVG_Image::resize ( int width, int height )
```

Have the svg data (re-)rasterized using the given width and height values.

By default, the resulting image w() and h() will be close to width and height while preserving the width/height ratio of the SVG data. If proportional was set to `false`, the image is rasterized to the exact width and height values. In both cases, data_w() and data_h() values are set to w() and h(), respectively.

12.136.4 Member Data Documentation

12.136.4.1 proportional

```cpp
bool Fl_SVG_Image::proportional
```

Set this to `false` to allow image re-scaling that alters the image aspect ratio. Upon object creation, proportional is set to `true`, and the aspect ratio is kept constant.

The documentation for this class was generated from the following files:

- Fl_SVG_Image.H
- Fl_SVG_Image.cxx

12.137 Fl_Sys_Menu_Bar Class Reference

A class to create and modify menus that appear on macOS in the menu bar at the top of the screen.

```cpp
#include <Fl_Sys_Menu_Bar.H>
```

Inheritance diagram for Fl_Sys_Menu_Bar:
Public Types

- enum window_menu_style_enum { no_window_menu = 0, tabbing_mode_none, tabbing_mode_automatic, tabbing_mode_preferred }

  Possible styles of the Window menu in the system menu bar.

Public Member Functions

- int add (const char *label, const char *shortcut, Fl_Callback *cb, void *user_data=0, int flags=0)
  Adds a new menu item.

- int add (const char *label, int shortcut, Fl_Callback *, void *user_data=0, int flags=0)
  Add a new menu item to the system menu bar.

- int add (const char *str)
  Forms-compatible procedure to add items to the system menu bar.

- void clear ()
  Set the Fl_Menu_Item array pointer to null, indicating a zero-length menu.

- int clear_submenu (int index)
  Clears the specified submenu pointed to by index of all menu items.

- Fl_Sys_Menu_Bar (int x, int y, int w, int h, const char *l=0)
  The constructor.

- int insert (int index, const char *label, const char *shortcut, Fl_Callback *cb, void *user_data=0, int flags=0)
  Insert a new menu item.

- int insert (int index, const char *label, int shortcut, Fl_Callback *cb, void *user_data=0, int flags=0)
  insert in the system menu bar a new menu item

- const Fl_Menu_Item *menu () const
  Return the system menu's array of Fl_Menu_Item's.

- void menu (const Fl_Menu_Item *m)
  create a system menu bar using the given list of menu structs

- int mode (int i) const
  Gets the flags of item i.

- void mode (int i, int fl)
  Sets the flags of item i.

- void remove (int n)
  remove an item from the system menu bar

- void replace (int index, const char *name)
  rename an item from the system menu bar

- void setonly (Fl_Menu_Item *item)
  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.

- void shortcut (int i, int s)
  Changes the shortcut of item i to n.

- void update () FL_OVERRIDE
  Updates the menu bar after any change to its items.

- virtual ~Fl_Sys_Menu_Bar ()
  The destructor.
Public Member Functions inherited from Fl_Menu_Bar

- **Fl_Menu_Bar**(int X, int Y, int W, int H, const char *l=0)
  Creates a new Fl_Menu_Bar widget using the given position, size, and label string.

- **int handle**(int) FL_OVERRIDE
  Handles the specified event.

Public Member Functions inherited from Fl_Menu_

- **int add**(const char *)
  This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|' separating the menu items, and tab separating the menu item names from an optional shortcut string.

- **int add**(const char *, int shortcut, Fl_Callback *, void *d=0, int e=0)
  Adds a new menu item.

- **int add**(const char *a, const char *b, Fl_Callback *c, void *d=0, int e=0)
  See int Fl_Menu_::add(const char* label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)

- **void clear()**
  Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.

- **int clear_submenu**(int index)
  Clears the specified submenu pointed to by index of all menu items.

- **void copy**(const Fl_Menu_Item *, void *user_data=0)
  Sets the menu array pointer with a copy of m that will be automatically deleted.

- **Fl_Boxtype down_box()** const
  This box type is used to surround the currently-selected items in the menus.

- **void down_box**(Fl_Boxtype b)
  Sets the box type used to surround the currently-selected items in the menus.

- **Fl_Color down_color()** const
  For back compatibility, same as selection_color()

- **void down_color**(unsigned c)
  For back compatibility, same as selection_color()

- **int find_index**(const char *) const
  Find the menu item index for a given menu pathname, such as “Edit/Copy”.

- **int find_index**(const Fl_Menu_Item *) const
  Find the index into the menu array for a given item.

- **int find_index**(Fl_Callback *) const
  Find the index into the menu array for a given callback cb.

- **const Fl_Menu_Item * find_item**(const char *)
  Find the menu item for a given menu pathname, such as “Edit/Copy”.

- **const Fl_Menu_Item * find_item**(Fl_Callback *)
  Find the menu item for the given callback cb.

- **const Fl_Menu_Item * find_item_with_argument**(long)
  Find the menu item for the given user argument v.

- **const Fl_Menu_Item * find_item_with_user_data**(void *)
  Find the menu item for the given user data v.

- **Fl_Menu_**(int, int, int, int, const char *)
  Creates a new Fl_Menu_ widget using the given position, size, and label string.

- **void global()**
  Make the shortcuts for this menu work no matter what window has the focus when you type it.

- **int insert**(int index, const char *, int shortcut, Fl_Callback *, void *=0, int e=0)
  Inserts a new menu item at the specified index position.

- **int insert**(int index, const char *a, const char *b, Fl_Callback *c, void *=0, int e=0)
  See int Fl_Menu_::insert(const char* label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)
• int item_pathname (char *name, int namelen, const Fl_Menu_Item *finditem=0) const
  Get the menu 'pathname' for the specified menuitem.
• const Fl_Menu_Item * menu () const
  Returns a pointer to the array of Fl_Menu_Items.
• void menu (const Fl_Menu_Item *m)
  Sets the menu array pointer directly.
• Fl_Boxtype menu_box () const
  Get the box type for the menu popup windows.
• void menu_box (Fl_Boxtype b)
  Set the box type for the menu popup windows.
• const Fl_Menu_Item * menu_end ()
  Finishes menu modifications and returns menu().
• int mode (int i) const
  Gets the flags of item i.
• void mode (int i, int fl)
  Sets the flags of item i.
• const Fl_Menu_Item * mvalue () const
  Returns a pointer to the last menu item that was picked.
• const Fl_Menu_Item * picked (const Fl_Menu_Item *)
  When user picks a menu item, call this.
• const Fl_Menu_Item * prev_mvalue () const
  Returns a pointer to the menu item that was picked before the current one was picked.
• void remove (int)
  Deletes item i from the menu.
• void replace (int, const char *)
  Changes the text of item i.
• void setonly (Fl_Menu_Item *item)
  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.
• void shortcut (int i, int s)
  Changes the shortcut of item i to s.
• int size () const
  This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.
• void size (int W, int H)
• const Fl_Menu_Item * test_shortcut ()
  Returns the menu item with the entered shortcut (key value).
• const char * text () const
  Returns the title of the last item chosen.
• const char * text (int i) const
  Returns the title of item i.
• Fl_Color textcolor () const
  Get the current color of menu item labels.
• void textcolor (Fl_Color c)
  Sets the current color of menu item labels.
• Fl_Font textfont () const
  Gets the current font of menu item labels.
• void textfont (Fl_Font c)
  Sets the current font of menu item labels.
• Fl_Fontsize textsize () const
  Gets the font size of menu item labels.
• void textsize (Fl_Fontsize c)
  Sets the font size of menu item labels.
• int value () const  
  Returns the index into menu() of the last item chosen by the user.
• int value (const Fl_Menu_Item ∗)  
  The value is the index into menu() of the last item chosen by the user.
• int value (int i)  
  The value is the index into menu() of the last item chosen by the user.

Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()  
  Activates the widget.
• unsigned int active () const  
  Returns whether the widget is active.
• int active_r () const  
  Returns whether the widget and all of its parents are active.
• Fl_Align align () const  
  Gets the label alignment.
• void align (Fl_Align alignment)  
  Sets the label alignment.
• long argument () const  
  Gets the current user data (long) argument that is passed to the callback function.
• void argument (long v)  
  Sets the current user data (long) argument that is passed to the callback function.
• virtual class Fl_Gl_Window ∗ as_gl_window ()  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const ∗ as_gl_window () const  
• virtual Fl_Group ∗ as_group ()  
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• virtual Fl_Group const ∗ as_group () const  
• virtual Fl_Window ∗ as_window ()  
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const ∗ as_window () const  
• void bind_deimage (Fl_Image ∗img)  
  Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage (int f)  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image (Fl_Image ∗img)  
  Sets the image to use as part of the widget label when in the active state.
• void bind_image (int f)  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• Fl_Boxtype box () const  
  Gets the box type of the widget.
• void box (Fl_Boxtype new_box)  
  Sets the box type for the widget.
• Fl_Callback_p callback () const  
  Gets the current callback function for the widget.
• void callback (Fl_Callback ∗cb)  
  Sets the current callback function for the widget.
• void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)
Sets the current callback function and managed user data for the widget.

- `void callback (Fl_Callback ∗cb, void ∗p)`
  Sets the current callback function and data for the widget.

- `void callback (Fl_Callback0 ∗cb)`
  Sets the current callback function for the widget.

- `void callback (Fl_Callback1 ∗cb, long p=0)`
  Sets the current callback function for the widget.

- `unsigned int changed () const`
  Checks if the widget value changed since the last callback.

- `void clear_active ()`
  Marks the widget as inactive without sending events or changing focus.

- `void clear_changed ()`
  Marks the value of the widget as unchanged.

- `void clear_damage (uchar c=0)`
  Clears or sets the damage flags.

- `void clear_output ()`
  Sets a widget to accept input.

- `void clear_visible ()`
  Hides the widget.

- `void clear_visible_focus ()`
  Disables keyboard focus navigation with this widget.

- `Fl_Color color () const`
  Gets the background color of the widget.

- `void color (Fl_Color bg)`
  Sets the background color of the widget.

- `void color (Fl_Color bg, Fl_Color sel)`
  Sets the background and selection color of the widget.

- `Fl_Color color2 () const`
  For back compatibility only.

- `void color2 (unsigned a)`
  For back compatibility only.

- `int contains (const Fl_Widget ∗w) const`
  Checks if w is a child of this widget.

- `void copy_label (const char ∗new_label)`
  Sets the current label.

- `void copy_tooltip (const char ∗text)`
  Sets the current tooltip text.

- `uchar damage () const`
  Returns non-zero if `draw()` needs to be called.

- `void damage (uchar c)`
  Sets the damage bits for the widget.

- `void damage (uchar c, int x, int y, int w, int h)`
  Sets the damage bits for an area inside the widget.

- `int damage_resize (int, int, int, int)`
  Internal use only.

- `void deactivate ()`
  Deactivates the widget.

- `Fl_Image ∗deimage ()`
  Gets the image that is used as part of the widget label when in the inactive state.

- `const Fl_Image ∗deimage () const`
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *wgt, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *wgt, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  ...
Gets the label type.

- **void labeltype (Fl_Labeltype a)**
  
  Sets the label type.

- **void measure_label (int &ww, int &hh) const**
  
  Sets width `ww` and height `hh` accordingly with the label size.

- **bool needs_keyboard () const**
  
  Returns whether this widget needs a keyboard.

- **void needs_keyboard (bool needs)**
  
  Sets whether this widget needs a keyboard.

- **unsigned int output () const**
  
  Returns if a widget is used for output only.

- **Fl_Group * parent () const**
  
  Returns a pointer to the parent widget.

- **void parent (Fl_Group *p)**
  
  Internal use only - "for hacks only".

- **void position (int X, int Y)**
  
  Repositions the window or widget.

- **void redraw ()**
  
  Schedules the drawing of the widget.

- **void redraw_label ()**
  
  Schedules the drawing of the label.

- **virtual void resize (int x, int y, int w, int h)**
  
  Changes the size or position of the widget.

- **Fl_Color selection_color () const**
  
  Gets the selection color.

- **void selection_color (Fl_Color a)**
  
  Sets the selection color.

- **void set_active ()**
  
  Marks the widget as active without sending events or changing focus.

- **void set_changed ()**
  
  Marks the value of the widget as changed.

- **void set_output ()**
  
  Sets a widget to output only.

- **void set_visible ()**
  
  Makes the widget visible.

- **void set_visible_focus ()**
  
  Enables keyboard focus navigation with this widget.

- **int shortcut_label () const**
  
  Returns whether the widget's label uses ' Pract to indicate shortcuts.

- **void shortcut_label (int value)**
  
  Sets whether the widget's label uses ' Pract to indicate shortcuts.

- **virtual void show ()**
  
  Makes a widget visible.

- **void size (int W, int H)**
  
  Changes the size of the widget.

- **int take_focus ()**
  
  Gives the widget the keyboard focus.

- **unsigned int takesevents () const**
  
  Returns if the widget is able to take events.

- **int test_shortcut ()**
  
  Returns true if the widget's label contains the entered ' Pract shortcut.
• **const char** ∗ *tooltip () const*
  
  *Gets the current tooltip text.*

• **void tooltip (const char ∗text)**

  *Sets the current tooltip text.*

• **Fl_Window ∗ top_window () const**

  *Returns a pointer to the top-level window for the widget.*

• **Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const**

  *Finds the x/y offset of the current widget relative to the top-level window.*

• **uchar type () const**

  *Gets the widget type.*

• **void type (uchar t)**

  *Sets the widget type.*

• **int use_accents_menu ()**

  *Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.*

• **void ∗ user_data () const**

  *Gets the user data for this widget.*

• **void user_data (Fl_Callback_User_Data ∗v, bool auto_free)**

  *Sets the user data for this widget.*

• **void user_data (void ∗v)**

  *Sets the user data for this widget.*

• **unsigned int visible () const**

  *Returns whether a widget is visible.*

• **unsigned int visible_focus () const**

  *Checks whether this widget has a visible focus.*

• **void visible_focus (int v)**

  *Modifies keyboard focus navigation.*

• **int visible_r () const**

  *Returns whether a widget and all its parents are visible.*

• **int w () const**

  *Gets the widget width.*

• **Fl_When when () const**

  *Returns the conditions under which the callback is called.*

• **void when (uchar i)**

  *Sets the flags used to decide when a callback is called.*

• **Fl_Window ∗ window () const**

  *Returns a pointer to the nearest parent window up the widget hierarchy.*

• **int x () const**

  *Gets the widget position in its window.*

• **int y () const**

  *Gets the widget position in its window.*

• **virtual ∼Fl_Widget ()**

  *Destroys the widget.*

**Static Public Member Functions**

• **static void about (Fl_Callback ∗cb, void ∗data)**

  *Attaches a callback to the "About myprog" item of the system application menu.*

• **static void create_window_menu ()**

  *Adds a Window menu, to the end of the system menu bar.*

• **static window_menu_style_enum window_menu_style ()**

  *Get the style of the Window menu in the system menu bar.*

• **static void window_menu_style (window_menu_style_enum style)**

  *Set the desired style of the Window menu in the system menu bar.*
Static Public Member Functions inherited from Fl_Widget

• static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
• static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
• static int test_shortcut (const char *t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Member Functions

• void draw () FL_OVERRIDE
  Draws the widget.

Protected Member Functions inherited from Fl_Menu_Item

• int item_pathname_ (char *name, int namelen, const Fl_Menu_Item *finditem, const Fl_Menu_Item *menu=0) const

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const
  Draws the widget box according its box style.
• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
• void draw_focus () const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
• void draw_label () const
  Draws the widget's label at the defined label position.
• void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
• Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.
• unsigned int flags () const
  Gets the widget flags mask.
• void h (int v)
  Internal use only.
• void set_flag (unsigned int c)
  Sets a flag in the flags mask.
• void w (int v)
  Internal use only.
• void x (int v)
  Internal use only.
• void y (int v)
  Internal use only.
Additional Inherited Members

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6,
  CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10,
  CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14,
  NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18,
  MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<25, USERFLAG2 = 1<<26,
  USERFLAG1 = 1<<27,
} flags possible values enumeration.

Protected Attributes inherited from Fl_Menu_

- uchar alloc
- uchar down_box_
- Fl_Boxtype menu_box_
- Fl_Color textcolor_
- Fl_Font textfont_
- Fl_Fontsize textsize_

12.137.1 Detailed Description

A class to create and modify menus that appear on macOS in the menu bar at the top of the screen.

On other than macOS platforms, Fl_Sys_Menu_Bar is a synonym of class Fl_Menu_Bar.

On the macOS platform, replace Fl_Menu_Bar with Fl_Sys_Menu_Bar, and a system menu at the top of the screen will be available. This menu will match an array of Fl_Menu_Item's exactly as in all other FLTK menus (except for the submenu with the application's own name and the 'Window' menu; see below). There is, though, an important difference between an Fl_Sys_Menu_Bar object under macOS and under other platforms: only a single object from this class can be created, because macOS uses a single system menu bar. Therefore, porting to macOS an app that creates, on other platforms, several Fl_Menu_Bar objects, one for each of several windows, is more complex than just replacing Fl_Menu_Bar by Fl_Sys_Menu_Bar.

On the macOS platform, the system menu bar of any FLTK app begins with the Application menu which the FLTK library automatically constructs. Functions Fl_Mac_App_Menu::custom_application_menu_items() and Fl_Sys_Menu_Bar::about() can be used to further customize the Application menu. The FLTK library also automatically constructs and handles a Window menu which can be further customized (or even removed) calling Fl_Sys_Menu_Bar::window_menu_style(window_menu_style_enum style). Other member functions of this class allow the app to generate the rest of the system menu bar. It is recommended to localize the system menu bar using the standard Mac OS X localization procedure (see Internationalization).

Changes to the menu state are immediately visible in the menubar when they are made using member functions of the Fl_Sys_Menu_Bar class. Other changes (e.g., by a call to Fl_Menu_Item::set()) should be followed by a call to update() to be visible in the menubar across all platforms. macOS global variable fl_sys_menu_bar points to the unique, current system menu bar.

A few FLTK menu features are not supported by the Mac System menu:

- no symbolic labels
- no embossed labels
- no font sizes

As described above, the submenu with the application's own name (usually the second submenu from the left, immediately following the "Apple" submenu) is a special case, and can be managed with
**Fl_Mac_App_Menu::custom_application_menu_items().** For example, to make your own "Appname -> Preferences" dialog, you might use:

```c
#include <FL/platform.H> // for Fl_Mac_App_Menu class
#include <FL/Fl_Sys_Menu_Bar.H> // for Fl_Menu_Item

void prefs_cb(Fl_Widget *w, void *data) {
    // ..Open your preferences dialog here..
}

int main(..) {
    // Items to add to the application menu
    static Fl_Menu_Item appitems[] = {
        { "Preferences", 0, prefs_cb, 0, 0 },
        { 0 }, { 0 }
    };
    Fl_Mac_App_Menu::custom_application_menu_items(appitems); // adds it
}
```

..the result being:

![Figure 12.46 Mac Application submenu](image)

### 12.137.2 Member Enumeration Documentation

#### 12.137.2.1 window_menu_style_enum

define Fl_Sys_Menu_Bar::window_menu_style_enum
Possible styles of the Window menu in the system menu bar.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_window_menu</td>
<td>No Window menu in the system menu bar.</td>
</tr>
<tr>
<td>tabbing_mode_none</td>
<td>No tabbed windows, but the system menu bar contains a Window menu.</td>
</tr>
<tr>
<td>tabbing_mode_automatic</td>
<td>Windows are created by themselves but can be tabbed later.</td>
</tr>
<tr>
<td>tabbing_mode_preferred</td>
<td>Windows are tabbed when created.</td>
</tr>
</tbody>
</table>

### 12.137.3 Constructor & Destructor Documentation

#### 12.137.3.1 Fl_Sys_Menu_Bar()

```c
Fl_Sys_Menu_Bar::Fl_Sys_Menu_Bar (int x, int y, int w, int h, const char *l = 0 )
```

The constructor.
On Mac OS X, all arguments are unused. On other platforms they are used as by `Fl_Menu_Bar::Fl_Menu_Bar()`.

### 12.137.4 Member Function Documentation

#### 12.137.4.1 about()

```c
void Fl_Sys_Menu_Bar::about (Fl_Callback * cb,`
```
void * data ) [static]
Attaches a callback to the "About myprog" item of the system application menu.
This cross-platform function is effective only under the MacOS platform.

Parameters

| cb    | a callback that will be called by "About myprog" menu item with NULL 1st argument. |
| data  | a pointer transmitted as 2nd argument to the callback. |

12.137.4.2  add() [1/3]

int Fl_Sys_Menu_Bar::add (  
    const char * label,  
    const char * shortcut,  
    Fl_Callback * cb,  
    void * user_data = 0,  
    int flags = 0 ) [inline]

Adds a new menu item.

See also

Fl_Menu_::add(const char* label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)

12.137.4.3  add() [2/3]

int Fl_Sys_Menu_Bar::add (  
    const char * label,  
    int shortcut,  
    Fl_Callback * cb,  
    void * user_data = 0,  
    int flags = 0 )

Add a new menu item to the system menu bar.
Add to the system menu bar a new menu item, with a title string, shortcut int, callback, argument to the callback, and flags.

Parameters

| label     | - new menu item's label |
| shortcut  | - new menu item's integer shortcut (can be 0 for none, or e.g. FL_ALT+'x') |
| cb        | - callback to be invoked when item selected (can be 0 for none, in which case the menubar's callback() can be used instead) |
| user_data | - argument to the callback |
| flags     | - item's flags, e.g. FL_MENU_TOGGLE, etc. |

Returns

the index into the menu() array, where the entry was added

See also

Fl_Menu_::add(const char* label, int shortcut, Fl_Callback *cb, void *user_data, int flags)

12.137.4.4  add() [3/3]

int Fl_Sys_Menu_Bar::add (  
    const char * str )

Forms-compatible procedure to add items to the system menu bar.
Returns
the index into the menu() array, where the entry was added

See also
Fl_Menu_::add(const char * str)

12.137.4.5 clear()
void Fl_Sys_Menu_Bar::clear ( )
Set the Fl_Menu_Item array pointer to null, indicating a zero-length menu.

See also
Fl_Menu_::clear()

12.137.4.6 clear_submenu()
int Fl_Sys_Menu_Bar::clear_submenu ( int index )
Clears the specified submenu pointed to by index of all menu items.

See also
Fl_Menu_::clear_submenu(int index)

12.137.4.7 create_window_menu()
void Fl_Sys_Menu_Bar::create_window_menu ( ) [static]
Adds a Window menu, to the end of the system menu bar.
FLTK apps typically don't need to call this function which is automatically called by the library the first time a window is shown. The default system menu bar contains a Window menu with a "Merge All Windows" item. Other Window menu styles can be obtained calling Fl_Sys_Menu_Bar::window_menu_style(window_menu_style_enum) before the first Fl_Window::show(). Alternatively, an app can call create_window_menu() after having populated the system menu bar, for example with menu(const Fl_Menu_Item *), and before the first Fl_Window::show(). This function does nothing on non MacOS platforms.

Version
1.4

12.137.4.8 draw()
void Fl_Sys_Menu_Bar::draw ( ) [protected], [virtual]
Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
Reimplemented from Fl_Menu_Bar.

12.137.4.9 insert() [1/2]
int Fl_Sys_Menu_Bar::insert ( int index,
const char * label,
const char * shortcut,
FL_Callback * cb,
void * user_data = 0,
int flags = 0) [inline]

Insert a new menu item.

See also

Fl_Menu_::insert(int index, const char * label, const char * shortcut, FL_Callback *cb, void *user_data=0, int flags=0)

12.137.4.10 insert() [2/2]

int Fl_Sys_Menu_Bar::insert (
    int index,
    const char * label,
    int shortcut,
    FL_Callback * cb,
    void * user_data = 0,
    int flags = 0 )

insert in the system menu bar a new menu item

Insert in the system menu bar a new menu item, with a title string, shortcut int, callback, argument to the callback, and flags.

Returns

the index into the menu() array, where the entry was inserted

See also

Fl_Menu_::insert(int index, const char * label, int shortcut, FL_Callback *cb, void *user_data, int flags)

12.137.4.11 menu()

void Fl_Sys_Menu_Bar::menu (
    const Fl_Menu_Item * m )

create a system menu bar using the given list of menu structs

Author

Matthias Melcher

Parameters

m | Zero-ending list of Fl_Menu_Item's

12.137.4.12 mode()

void Fl_Sys_Menu_Bar::mode ( 
    int i,
    int fl )

Sets the flags of item i.
See also

`Fl_Menu_::mode(int i, int fl)`

12.137.4.13 remove()

void Fl_Sys_Menu_Bar::remove (int index)

remove an item from the system menu bar

Parameters

| index | the index of the item to remove |

12.137.4.14 replace()

void Fl_Sys_Menu_Bar::replace (int index, const char * name)

rename an item from the system menu bar

Parameters

| index | the index of the item to rename |
| name  | the new item name as a UTF8 string |

12.137.4.15 update()

void Fl_Sys_Menu_Bar::update ( ) [virtual]

Updates the menu bar after any change to its items.

This is useful when the menu bar can be an Fl_Sys_Menu_Bar object.

Reimplemented from Fl_Menu_Bar.

12.137.4.16 window_menu_style()

void Fl_Sys_Menu_Bar::window_menu_style (Fl_Sys_Menu_Bar::window_menu_style_enum style ) [static]

Set the desired style of the Window menu in the system menu bar.

This function, to be called before the first call to Fl_Window::show(), allows to control whether the system menu bar should contain a Window menu, and if yes, whether new windows should be displayed in tabbed form. These are the effects of various values for style:

- no_window_menu: don't add a Window menu to the system menu bar
- tabbing_mode_none: add a simple Window menu to the system menu bar
- tabbing_mode_automatice: the window menu also contains "Merge All Windows" to group all windows in a single tabbed display mode. This is the default Window menu style for FLTK apps.
- tabbing_mode_preferred: new windows are displayed in tabbed mode when first created

The Window menu, if present, is entirely created and controlled by the FLTK library. Mac OS version 10.12 or later must be running for windows to be displayed in tabbed form. Under non MacOS platforms, this function does nothing.
The documentation for this class was generated from the following files:

- Fl_Sys_Menu_Bar.H
- Fl_Sys_Menu_Bar.cxx

12.138 Fl_Table Class Reference

A table of widgets or other content.

#include <Fl_Table.H>

Inheritance diagram for Fl_Table:

```
Fl_Widget
    └── Fl_Group
        └── Fl_Table
            └── Fl_Table_Row
```

**Public Types**

- enum TableContext {
  CONTEXT_NONE = 0, CONTEXT_STARTPAGE = 0x01, CONTEXT_ENDPAGE = 0x02, CONTEXT_ROW_HEADER = 0x04,
  CONTEXT_COL_HEADER = 0x08, CONTEXT_CELL = 0x10, CONTEXT_TABLE = 0x20, CONTEXT_RC_RESIZE = 0x40 }

  The context bit flags for Fl_Table related callbacks.

**Public Member Functions**

- void add (Fl_Widget &wgt)
  The specified widget is removed from its current group (if any) and added to the end of Fl_Table's group.
- void add (Fl_Widget *wgt)
  The specified widget is removed from its current group (if any) and added to the end of Fl_Table's group.
- Fl_Widget *const *array ()
  Returns a pointer to the array of children.
- int children () const
  Returns the number of children in the table.
- int callback_col ()
  Returns the current column the event occurred on.
- int callback_context ()
  Returns the current ‘table context’.
- int callback_row ()
  Returns the current row the event occurred on.
- Fl_Widget *child (int n) const
  Returns the child widget by an index.
- void callback (Fl_Widget * , void * )
  Callbacks will be called depending on the setting of Fl_Widget::when().
• virtual void clear ()
  Clears the table to zero rows (rows(0)), zero columns (cols(0)), and clears any widgets (table->clear()) that were added with begin()/end() or add()/insert()/etc.

• int col_header ()
  Returns if column headers are enabled or not.

• void col_header (int flag)
  Enable or disable column headers.

• int col_header_height ()
  Returns the column header height.

• void col_header_height (int height)
  Sets the height in pixels for column headers and redraws the table.

• int col_position ()
  Returns the current column scroll position as a column number.

• void col_position (int col)
  Sets the horizontal scroll position so 'col' is at the left, and causes the screen to redraw.

• int col_resize ()
  Returns if column resizing by the user is allowed.

• void col_resize (int flag)
  Allows/disallows column resizing by the user.

• int col_resize_min ()
  Returns the current column minimum resize value.

• void col_resize_min (int val)
  Sets the current column minimum resize value.

• int col_width (int col)
  Returns the current width of the specified column in pixels.

• void col_width (int col, int width)
  Sets the width of the specified column in pixels, and the table is redrawn.

• void col_width_all (int width)
  Convenience method to set the width of all columns to the same value, in pixels.

• int cols ()
  Get the number of columns in the table.

• virtual void cols (int val)
  Set the number of columns in the table and redraw.

• void do_callback (TableContext context, int row, int col)
  Calls the widget callback.

• void end ()
• int find (const Fl_Widget &wgt) const
• int find (const Fl_Widget ∗wgt) const

• Fl_Table (int X, int Y, int W, int H, const char ∗l=0)
  The constructor for Fl_Table.

• void get_selection (int &row_top, int &col_left, int &row_bot, int &col_right)
  Gets the region of cells selected (highlighted).

• void init_sizes ()
  Resets the internal array of widget sizes and positions.

• void insert (Fl_Widget &wgt, Fl_Widget ∗wgt)
  The specified widget is removed from its current group (if any) and inserted into Fl_Table's group before widget 'w2'.

• void insert (Fl_Widget &wgt, int n)
The specified widget is removed from its current group (if any) and inserted into the Fl_Table's group at position 'n'.

- **int is_interactive_resize ()**
  Returns 1 if someone is interactively resizing a row or column.

- **int is_selected (int r, int c)**
  See if the cell at row `r` and column `c` is selected.

- **int move_cursor (int R, int C)**
  Same as move_cursor(R,C,1);.

- **int move_cursor (int R, int C, int shiftselect)**
  Moves the selection cursor a relative number of rows/columns specified by R/C.

- **void remove (Fl_Widget &wgt)**
  The specified widget is removed from Fl_Table’s group.

- **void resize (int X, int Y, int W, int H) FL_OVERRIDE**
  Handle resize events if user resizes parent window.

- **int row_header ()**
  Returns if row headers are enabled or not.

- **void row_header (int flag)**
  Enables/disables showing the row headers.

- **Fl_Color row_header_color ()**
  Returns the current row header color.

- **void row_header_color (Fl_Color val)**
  Sets the row header color and causes the screen to redraw.

- **int row_header_width ()**
  Returns the current row header width (in pixels).

- **void row_header_width (int width)**
  Sets the row header width to n and causes the screen to redraw.

- **int row_height (int row)**
  Returns the current height of the specified row as a value in pixels.

- **void row_height (int row, int height)**
  Sets the height of the specified row in pixels, and the table is redrawn.

- **void row_height_all (int height)**
  Convenience method to set the height of all rows to the same value, in pixels.

- **int row_position ()**
  Returns the current row scroll position as a row number.

- **void row_position (int row)**
  Sets the vertical scroll position so 'row' is at the top, and causes the screen to redraw.

- **int row_resize ()**
  Returns if row resizing by the user is allowed.

- **void row_resize (int flag)**
  Allows/disallows row resizing by the user.

- **int row_resize_min ()**
  Returns the current row minimum resize value.

- **void row_resize_min (int val)**
  Sets the current row minimum resize value.

- **int rows ()**
  Returns the number of rows in the table.

- **virtual void rows (int val)**
  Sets the number of rows in the table, and the table is redrawn.

- **int scrollbar_size () const**
  Gets the current size of the scrollbars’ troughs, in pixels.

- **void scrollbar_size (int newSize)**
  Sets the pixel size of the scrollbars’ troughs to newSize, in pixels.
• void set_selection (int row_top, int col_left, int row_bot, int col_right)
  Sets the region of cells to be selected (highlighted).

• int tab_cell_nav () const
  Get state of table's 'Tab' key cell navigation flag.

• void tab_cell_nav (int val)
  Flag to control if Tab navigates table cells or not.

• void table_box (Fl_Boxtype val)
  Sets the kind of box drawn around the data table, the default being FL_NO_BOX.

• Fl_Boxtype table_box (void)
  Returns the current box type used for the data table.

• void top_row ()
  Returns the current top row shown in the table.

• void top_row (int row)
  Sets which row should be at the top of the table, scrolling as necessary, and the table is redrawn.

• void visible_cells (int &r1, int &r2, int &c1, int &c2)
  Returns the range of row and column numbers for all visible and partially visible cells in the table.

• void when (Fl_When flags)
  The Fl_Widget::when() function is used to set a group of flags, determining when the widget callback is called:

• ~Fi_Table ()
  The destructor for Fl_Table.

Public Member Functions inherited from Fl_Group

• Fl_Widget const & _ddfdesign_kludge ()
  This is for forms compatibility only.

• void add (Fl_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.

• void add (Fl_Widget *o)
  See void Fl_Group::add(Fl_Widget &w)

• void add_resizable (Fl_Widget &o)
  Adds a widget to the group and makes it the resizable widget.

• Fl_Widget *const * array () const
  Returns a pointer to the array of children.

• Fl_Group const * as_group () const FL_OVERRIDE

• Fl_Group * as_group () FL_OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• void begin ()
  Sets the current group so you can build the widget tree by just constructing the widgets.

• Fl_Widget * child (int n) const
  Returns array()[n].

• int children () const
  Returns how many child widgets the group has.

• void clear ()
  Deletes all child widgets from memory recursively.

• unsigned int clip_children ()
  Returns the current clipping mode.

• void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

• virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.

• void end ()
Exactly the same as current(this->parent()).

- **int find (const Fl_Widget &o) const**
  
  See int Fl_Group::find(const Fl_Widget *w) const.

- **int find (const Fl_Widget *) const**
  
  Searches the child array for the widget and returns the index.

- **Fl_Group (int, int, int, int, const char * = 0)**
  
  Creates a new Fl_Group widget using the given position, size, and label string.

- **void focus (Fl_Widget *)**
  
  This is for forms compatibility only.

- **void init_sizes ()**
  
  Resets the internal array of widget sizes and positions.

- **void insert (Fl_Widget &, int i)**
  
  The widget is removed from its current group (if any) and then inserted into this group.

- **void insert (Fl_Widget &o, Fl_Widget *before)**
  
  This does insert(w, find(before)).

- **void remove (Fl_Widget &)**
  
  Removes a widget from the group but does not delete it.

- **void remove (Fl_Widget *o)**
  
  Removes the widget o from the group.

- **void remove (int index)**
  
  Removes the widget at index from the group but does not delete it.

- **Fl_Widget * resizable () const**
  
  Returns the group's resizable widget.

- **void resizable (Fl_Widget &o)**
  
  Sets the group's resizable widget.

- **void resizable (Fl_Widget *o)**
  
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- **virtual ~Fl_Group ()**
  
  The destructor also deletes all the children.

**Public Member Functions inherited from Fl_Widget**

- **void _clear_fullscreen ()**

- **void _set_fullscreen ()**

- **void activate ()**

  Activates the widget.

- **unsigned int active () const**

  Returns whether the widget is active.

- **int active_r () const**

  Returns whether the widget and all of its parents are active.

- **Fl_Align align () const**

  Gets the label alignment.

- **void align (Fl_Align alignment)**

  Sets the label alignment.

- **long argument () const**

  Gets the current user data (long) argument that is passed to the callback function.

- **void argument (long v)**

  Sets the current user data (long) argument that is passed to the callback function.

- **virtual class Fl_Gl_Window * as_gl_window ()**

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
virtual class Fl_Gl_Window const * as_gl_window () const
virtual Fl_Window * as_window ()

Returns an Fl_Window pointer if this widget is an Fl_Window.

virtual Fl_Window const * as_window () const

void bind_deimage (Fl_Image *img)

Sets the image to use as part of the widget label when in the inactive state.

void bind_deimage (int f)

Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

void bind_image (Fl_Image *img)

Sets the image to use as part of the widget label when in the active state.

void bind_image (int f)

Bind the image to the widget, so the widget will delete the image when it is no longer needed.

Fl_Boxtype box () const

Gets the box type of the widget.

void box (Fl_Boxtype new_box)

Sets the box type for the widget.

Fl_Callback_p callback () const

Gets the current callback function for the widget.

void callback (Fl_Callback *cb)

Sets the current callback function for the widget.

void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

Sets the current callback function and managed user data for the widget.

void callback (Fl_Callback *cb, void *p)

Sets the current callback function and data for the widget.

void callback (Fl_Callback0 *cb)

Sets the current callback function for the widget.

void callback (Fl_Callback1 *cb, long p=0)

Sets the current callback function for the widget.

unsigned int changed () const

Checks if the widget value changed since the last callback.

void clear_active ()

Marks the widget as inactive without sending events or changing focus.

void clear_changed ()

Marks the value of the widget as unchanged.

void clear_damage (uchar c=0)

Clears or sets the damage flags.

void clear_output ()

Sets a widget to accept input.

void clear_visible ()

Hides the widget.

void clear_visible_focus ()

Disables keyboard focus navigation with this widget.

Fl_Color color () const

Gets the background color of the widget.

void color (Fl_Color bg)

Sets the background color of the widget.

void color (Fl_Color bg, Fl_Color sel)

Sets the background and selection color of the widget.

Fl_Color color2 () const

For back compatibility only.

void color2 (unsigned a)
For back compatibility only.

- `int contains` (const `Fl_Widget` *w) const
  Checks if `w` is a child of this widget.

- `void copy_label` (const char *new_label)
  Sets the current label.

- `void copy_tooltip` (const char *text)
  Sets the current tooltip text.

- `uchar damage()` const
  Returns non-zero if `draw()` needs to be called.

- `void damage`(uchar c)
  Sets the damage bits for the widget.

- `void damage`(uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- `int damage_resize`(int, int, int, int)
  Internal use only.

- `void deactivate`
  Deactivates the widget.

- `Fl_Image *deimage`
  Gets the image that is used as part of the widget label when in the inactive state.

- `const Fl_Image *deimage` const
  Gets the image that is used as part of the widget label when in the inactive state.

- `void deimage`(Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

- `void deimage`(Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

- `int deimage_bound` const
  Returns whether the inactive image is managed by the widget.

- `void do_callback`(Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

- `void do_callback`(Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- `void do_callback`(Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- `void draw_label`(int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- `int h()` const
  Gets the widget height.

- `virtual void hide`
  Makes a widget invisible.

- `Fl_Image *image`
  Gets the image that is used as part of the widget label when in the active state.

- `const Fl_Image *image` const
  Gets the image that is used as part of the widget label when in the active state.

- `void image`(Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.

- `void image`(Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.

- `int image_bound` const
  Returns whether the image is managed by the widget.

- `int inside` (const `Fl_Widget` *wgt) const
  Checks if this widget is a child of `wgt`.
• int **is_label_copied** () const
  Returns whether the current label was assigned with **copy_label()**.
• const char **label** () const
  Gets the current label text.
• void **label** (const char **text**)
  Sets the current label pointer.
• void **label** (Fl_Labeltype a, const char **b**)
  Shortcut to set the label text and type in one call.
• Fl_Color **labelcolor** () const
  Gets the label color.
• void **labelcolor** (Fl_Color c)
  Sets the label color.
• Fl_Font **labelfont** () const
  Gets the font to use.
• void **labelfont** (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize **labelsize** () const
  Gets the font size in pixels.
• void **labelsize** (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype **labeltype** () const
  Gets the label type.
• void **labeltype** (Fl_Labeltype a)
  Sets the label type.
• void **measure_label** (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool **needs_keyboard** () const
  Returns whether this widget needs a keyboard.
• void **needs_keyboard** (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int **output** () const
  Returns if a widget is used for output only.
• Fl_Group **parent** () const
  Returns a pointer to the parent widget.
• void **parent** (Fl_Group **p**)
  Internal use only - "for hacks only".
• void **position** (int X, int Y)
  Repositions the window or widget.
• void **redraw** ()
  Schedules the drawing of the widget.
• void **redraw_label** ()
  Schedules the drawing of the label.
• Fl_Color **selection_color** () const
  Gets the selection color.
• void **selection_color** (Fl_Color a)
  Sets the selection color.
• void **set_active** ()
  Marks the widget as active without sending events or changing focus.
• void **set_changed** ()
  Marks the value of the widget as changed.
• void **set_output** ()
Sets a widget to output only.

- void **set_visible ()**
  
  Makes the widget visible.

- void **set_visible_focus ()**
  
  Enables keyboard focus navigation with this widget.

- int **shortcut_label () const**
  
  Returns whether the widget's label uses '&' to indicate shortcuts.

- void **shortcut_label (int value)**
  
  Sets whether the widget's label uses '&' to indicate shortcuts.

- virtual void **show ()**
  
  Makes a widget visible.

- void **size (int W, int H)**
  
  Changes the size of the widget.

- int **take_focus ()**
  
  Gives the widget the keyboard focus.

- unsigned int **takesevents () const**
  
  Returns if the widget is able to take events.

- int **test_shortcut ()**
  
  Returns true if the widget's label contains the entered 'x' shortcut.

- const char ∗ **tooltip () const**
  
  Gets the current tooltip text.

- void **tooltip (const char ∗text)**
  
  Sets the current tooltip text.

- Fl_Window ∗ **top_window () const**
  
  Returns a pointer to the top-level window for the widget.

- Fl_Window ∗ **top_window_offset (int &xoff, int &yoff) const**
  
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar **type () const**
  
  Gets the widget type.

- void **type (uchar t)**
  
  Sets the widget type.

- int **use_accents_menu ()**
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void ∗ **user_data () const**
  
  Gets the user data for this widget.

- void **user_data (Fl_Callback_User_Data ∗v, bool auto_free)**
  
  Sets the user data for this widget.

- void **user_data (void ∗v)**
  
  Sets the user data for this widget.

- unsigned int **visible () const**
  
  Returns whether a widget is visible.

- unsigned int **visible_focus () const**
  
  Checks whether this widget has a visible focus.

- void **visible_focus (int v)**
  
  Modifies keyboard focus navigation.

- int **visible_r () const**
  
  Returns whether a widget and all its parents are visible.

- int **w () const**
  
  Gets the widget width.

- Fl_When **when () const**
  
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window *window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.

Protected Types

• enum ResizeFlag
  RESIZE_NONE = 0 , RESIZE_COL_LEFT = 1 , RESIZE_COL_RIGHT = 2 , RESIZE_ROW_ABOVE = 3 ,
  RESIZE_ROW_BELOW = 4

Protected Types inherited from Fl_Widget

• enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31
}
  flags possible values enumeration.

Protected Member Functions

• void change_cursor (Fl_Cursor newcursor)
  Change mouse cursor to different type.
• long col_scroll_position (int col)
  Returns the scroll position (in pixels) of the specified column 'col'.
• TableContext cursor2rowcol (int &R, int &C, ResizeFlag &resizeflag)
  Find row/col for the recent mouse event.
• void damage_zone (int r1, int c1, int r2, int c2, int r3=0, int c3=0)
  Sets the damage zone to the specified row/col values.
• void draw () FL_OVERRIDE
  Draws the entire Fl_Table.
• virtual void draw_cell (TableContext context, int R=0, int C=0, int X=0, int Y=0, int W=0, int H=0)
  Subclass should override this method to handle drawing the cells.
• int find_cell (TableContext context, int R, int C, int &X, int &Y, int &W, int &H)
  Find a cell's X/Y/W/H region for the specified cell in row 'R', column 'C'.
• void get_bounds (TableContext context, int &X, int &Y, int &W, int &H)
  Returns the (X,Y,W,H) bounding region for the specified 'context'.
• int handle (int e) FL_OVERRIDE
  Handle FLTK events.
• int **is_fltk_container**
  
  Does the table contain any child fltk widgets?

• void **recalc_dimensions**
  
  Recalculate the dimensions of the table, and affect any children.

• void **redraw_range** (int topRow, int botRow, int leftCol, int rightCol)
  
  Define region of cells to be redrawn by specified range of rows/cols, and then sets damage(DAMAGE_CHILD).

• int **row_col_clamp** (TableContext context, int &R, int &C)
  
  Return specified row/col values R and C to within the table’s current row/col limits.

• long **row_scroll_position** (int row)
  
  Returns the scroll position (in pixels) of the specified ‘row’.

• void **table_resized**
  
  Call this if table was resized, to recalculate internal data.

• void **table_scrolled**
  
  Recalculate internals after a scroll.

Protected Member Functions inherited from Fl_Group

• Fl_Rect * **bounds**
  
  Returns the internal array of widget sizes and positions.

• void **draw_child** (Fl_Widget &widget) const
  
  Forces a child to redraw.

• void **draw_children**
  
  Draws all children of the group.

• void **draw_outside_label** (const Fl_Widget &widget) const
  
  Parents normally call this to draw outside labels of child widgets.

• virtual int **on_insert** (Fl_Widget *, int)
  
  Allow derived groups to act when a widget is added as a child.

• virtual int **on_move** (int, int)
  
  Allow derived groups to act when a widget is moved within the group.

• virtual void **on_remove** (int)
  
  Allow derived groups to act when a child widget is removed from the group.

• int * **sizes**
  
  Returns the internal array of widget sizes and positions.

• void **update_child** (Fl_Widget &widget) const
  
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

• void **clear_flag** (unsigned int c)
  
  Clears a flag in the flags mask.

• void **draw_backdrop** () const
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• void **draw_box** () const
  
  Draws the widget box according its box style.

• void **draw_box** (Fl_Boxtype t, Fl_Color c) const
  
  Draws a box of type t, of color c at the widget’s position and size.

• void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type t, of color c at the position X,Y and size W,H.

• void **draw_focus** () const
  
  Draws a focus rectangle around the widget.

• void **draw_focus** (Fl_Boxtype t, int X, int Y, int W, int H) const
Draws a focus rectangle around the widget.

- **void** draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.

- **void** draw_label () const
  
  Draws the widget's label at the defined label position.

- **void** draw_label (int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.

- **Fl_Widget** (int x, int y, int w, int h, const char ∗label=0L)
  
  Creates a widget at the given position and size.

- unsigned int flags () const
  
  Gets the widget flags mask.

- **void** h (int v)
  
  Internal use only.

- **void** set_flag (unsigned int c)
  
  Sets a flag in the flags mask.

- **void** w (int v)
  
  Internal use only.

- **void** x (int v)
  
  Internal use only.

- **void** y (int v)
  
  Internal use only.

Static Protected Member Functions

- static void scroll_cb (Fl_Widget ∗, void ∗)
  
  Callback for when someone moves a scrollbar.

Protected Attributes

- int botrow
  
  bottom row# of currently visible table on screen

- int current_col
  
  selection cursor's current column (-1 if none)

- int current_row
  
  selection cursor's current row (-1 if none)

- Fl_Scrollbar ∗ hscrollbar
  
  child horizontal scrollbar widget

- int leftcol
  
  left column# of currently visible table on screen

- int leftcol_scrollpos
  
  precomputed scroll position for left column

- int rightcol
  
  right column# of currently visible table on screen

- int select_col
  
  extended selection column (-1 if none)

- int select_row
  
  extended selection row (-1 if none)

- Fl_Scroll ∗ table
  
  child Fl_Scroll widget container for child fltk widgets (if any)

- int table_h
  
  table's virtual height (in pixels)

- int table_w
table's virtual width (in pixels)

- **int** tih
  Data table's inner h dimension, inside bounding box. See Table Dimension Diagram.

- **int** tiw
  Data table's inner w dimension, inside bounding box. See Table Dimension Diagram.

- **int** tix
  Data table's inner x dimension, inside bounding box. See Table Dimension Diagram.

- **int** tly
  Data table's inner y dimension, inside bounding box. See Table Dimension Diagram.

- **int** toh
  Data table's outer h dimension, outside bounding box. See Table Dimension Diagram.

- **int** toprow
  top row# of currently visible table on screen

- **int** toprow_scrollpos
  precomputed scroll position for top row

- **int** tow
  Data table's outer w dimension, outside bounding box. See Table Dimension Diagram.

- **int** tox
  Data table's outer x dimension, outside bounding box. See Table Dimension Diagram.

- **int** toy
  Data table's outer y dimension, outside bounding box. See Table Dimension Diagram.

- **Fl_Scrollbar** ∗ vscrollbar
  child vertical scrollbar widget

- **int** wiw
  Table widget's inner w dimension, inside bounding box. See Table Dimension Diagram.

- **int** wiw
  Table widget's inner w dimension, inside bounding box. See Table Dimension Diagram.

- **int** wix
  Table widget's inner x dimension, inside bounding box. See Table Dimension Diagram.

- **int** wiy
  Table widget's inner y dimension, inside bounding box. See Table Dimension Diagram.

Additional Inherited Members

Static Public Member Functions inherited from **Fl_Group**

- static Fl_Group ∗ current ()
  Returns the currently active group.

- static void current (Fl_Group ∗g)
  Sets the current group.

Static Public Member Functions inherited from **Fl_Widget**

- static void default_callback (Fl_Widget ∗widget, void ∗data)
  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char ∗t)
  Returns the Unicode value of the 'x' shortcut in a given text.

- static int test_shortcut (const char ∗, const bool require_alt=false)
  Returns true if the given text contains the entered 'x' shortcut.
12.138.1 Detailed Description

A table of widgets or other content.
This is the base class for table widgets.
To be useful it must be subclassed and several virtual functions defined. Normally applications use widgets derived from this widget, and do not use this widget directly; this widget is usually too low level to be used directly by applications.

This widget does not handle the data in the table. The `draw_cell()` method must be overridden by a subclass to manage drawing the contents of the cells.

This widget can be used in several ways:

- As a custom widget; see examples/table-simple.cxx and test/table.cxx. Very optimal for even extremely large tables.
- As a table made up of a single FLTK widget instanced all over the table, simulating a numeric spreadsheet. See examples/table-spreadsheet.cxx and examples/table-spreadsheet-with-keyboard-nav.cxx. Optimal for large tables.
- As a regular container of FLTK widgets, one widget per cell. See examples/table-as-container.cxx. Not recommended for large tables.

![Figure 12.47 table-simple example](image)

### Figure 12.47 table-simple example

![Figure 12.48 table-as-container example](image)

![Figure 12.48 table-as-container example](image)

When acting as part of a custom widget, events on the cells and/or headings generate callbacks when they are clicked by the user. You control when events are generated based on the setting for `Fl_Table::when()`.

When acting as a container for FLTK widgets, the FLTK widgets maintain themselves. Although the `draw_cell()` method must be overridden, its contents can be very simple. See the `draw_cell()` code in examples/table-simple.cxx.
The following variables are available to classes deriving from Fl_Table:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x()/y()/w()/h()</td>
<td>Fl_Table widget's outer dimension. The outer edge of the border of the Fl_Table. (Red in the diagram above)</td>
</tr>
<tr>
<td>wix/wiy/wiw/wih</td>
<td>Fl_Table widget's inner dimension. The inner edge of the border of the Fl_Table. eg. if the Fl_Table's box() is FL_NO_BOX, these values are the same as x()/y()/w()/h(). (Yellow in the diagram above)</td>
</tr>
<tr>
<td>tox/toy/tow/toh</td>
<td>The table's outer dimension. The outer edge of the border around the cells, but inside the row/col headings and scrollbars. (Green in the diagram above)</td>
</tr>
<tr>
<td>tix/tiy/tiw/tih</td>
<td>The table's inner dimension. The inner edge of the border around the cells, but inside the row/col headings and scrollbars. AKA the table's clip region. eg. if the table_box() is FL_NO_BOX, these values are the same as tox/toy/tow/toh. (Blue in the diagram above)</td>
</tr>
</tbody>
</table>

**CORE DEVELOPERS**
- Greg Ercolano : 12/16/2002 - initial implementation 12/16/02. Fl_Table, Fl_Table_Row, docs.
- Jean-Marc Lienher : 02/22/2004 - added keyboard nav + mouse selection, and ported Fl_Table into fltk-utf8-1.1.4

**OTHER CONTRIBUTORS**
- Inspired by the Feb 2000 version of FLVW's Flvw_Table widget. Mucho thanks to those folks.
- Mister Satan : 04/07/2003 - MinGW porting mods, and singleinput.cxx; a cool Fl_Input oriented spreadsheet example
- Marek Paliwoda : 01/08/2003 - Porting mods for Borland
- Ori Berger : 03/16/2006 - Optimizations for >500k rows/cols

**LICENSE**
Greg kindly gave his permission to integrate Fl_Table and Fl_Table_Row into FLTK, allowing FLTK license to apply while his widgets are part of the library. [updated by Greg, 04/26/17]

### 12.138.2 Member Enumeration Documentation

#### 12.138.2.1 TableContext

```cpp
enum Fl_Table::TableContext
```
The context bit flags for Fl_Table related callbacks. Should be used in draw_cell() to determine what’s being drawn, or in a callback() to determine where a recent event occurred.

**Enumerator**

<table>
<thead>
<tr>
<th>Enumerates</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTEXT_NONE</td>
<td>no known context</td>
</tr>
<tr>
<td>CONTEXT_STARTPAGE</td>
<td>before the table is redrawn</td>
</tr>
<tr>
<td>CONTEXT_ENDPAGE</td>
<td>after the table is redrawn</td>
</tr>
<tr>
<td>CONTEXT_ROW_HEADER</td>
<td>drawing or event occurred in the row header</td>
</tr>
<tr>
<td>CONTEXT_COL_HEADER</td>
<td>drawing or event occurred in the col header</td>
</tr>
<tr>
<td>CONTEXT_CELL</td>
<td>drawing or event occurred in a cell</td>
</tr>
<tr>
<td>CONTEXT_TABLE</td>
<td>drawing or event occurred in a dead zone of table</td>
</tr>
<tr>
<td>CONTEXT_RC_RESIZE</td>
<td>column or row is being resized</td>
</tr>
</tbody>
</table>

### 12.138.3 Constructor & Destructor Documentation

#### 12.138.3.1 Fl_Table()

```cpp
Fl_Table::Fl_Table (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char * l = 0 )
```

The constructor for Fl_Table.
This creates an empty table with no rows or columns, with headers and row/column resize behavior disabled.

#### 12.138.3.2 ~Fl_Table()

```cpp
Fl_Table::~Fl_Table ( )
```

The destructor for Fl_Table.
Destroys the table and its associated widgets.

### 12.138.4 Member Function Documentation

#### 12.138.4.1 array()

```cpp
Fl_Widget *const * Fl_Table::array ( ) [inline]
```

Returns a pointer to the array of children.
*This pointer is only valid until the next time a child is added or removed.*

#### 12.138.4.2 callback()

```cpp
void Fl_Table::callback (  
    Fl_Widget *,  
    void *)
```

Callbacks will be called depending on the setting of Fl_Widget::when().
Callback functions should use the following functions to determine the context/row/column:

- **Fl_Table::callback_row()** returns current row
- **Fl_Table::callback_col()** returns current column
- **Fl_Table::callback_context()** returns current table context

callback_row() and callback_col() will be set to the row and column number the event occurred on. If someone clicked on a row header, **row** will be 0. If someone clicked on a column header, **col** will be 0.
callback_context() will return one of the following:
### Fl_Table Class Reference

<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fl_Table::CONTEXT_ROW_HEADER</strong></td>
<td>Someone clicked on a row header. Excludes resizing.</td>
</tr>
<tr>
<td><strong>Fl_Table::CONTEXT_COL_HEADER</strong></td>
<td>Someone clicked on a column header. Excludes resizing.</td>
</tr>
<tr>
<td><strong>Fl_Table::CONTEXT_CELL</strong></td>
<td>Someone clicked on a cell. To receive callbacks for FL_RELEASE events, you must set when(FL_WHEN_RELEASE).</td>
</tr>
<tr>
<td><strong>Fl_Table::CONTEXT_RC_RESIZE</strong></td>
<td>Someone is resizing rows/columns either interactively, or via the <code>col_width()</code> or <code>row_height()</code> API. Use <code>is_interactive_resize()</code> to determine interactive resizing. If resizing a column, R=0 and C=column being resized. If resizing a row, C=0 and R=row being resized. NOTE: To receive resize events, you must set when(FL_WHEN_CHANGED).</td>
</tr>
</tbody>
</table>

```cpp
class MyTable : public Fl_Table {
    // Handle events that happen on the table
    void event_callback2() {
        int R = callback_row(), // row where event occurred
             C = callback_col(); // column where event occurred
        TableContext context = callback_context(); // which part of table
        fprintf(stderr, "callback: Row=%d Col=%d Context=%d Event=%d\n",
                R, C, (int)context, (int)Fl::event());
    }

    // Actual static callback
    static void event_callback(Fl_Widget*, void* data) {
        MyTable *o = (MyTable*)data;
        o->event_callback2();
    }

public:
    // Constructor
    MyTable() {
        table.callback(&event_callback, (void*)this); // setup callback
        table.when(FL_WHEN_CHANGED|FL_WHEN_RELEASE); // when to call it
    }
};
```

#### 12.138.4.3 callback_col()

int Fl_Table::callback_col () [inline]

Returns the current column the event occurred on.
This function should only be used from within the user's callback function.

#### 12.138.4.4 callback_context()

TableContext Fl_Table::callback_context () [inline]

Returns the current 'table context'.
This function should only be used from within the user's callback function.

#### 12.138.4.5 callback_row()

int Fl_Table::callback_row () [inline]

Returns the current row the event occurred on.
This function should only be used from within the user's callback function.

#### 12.138.4.6 child()

Fl_Widget * Fl_Table::child (int n) const [inline]

Returns the child widget by an index.
When using the Fl_Table as a container for FLTK widgets, this method returns the widget pointer from the internal array of widgets in the container.
Typically used in loops, eg:
for ( int i=0; i<children(); i++ ) {
    Fl_Widget *w = child(i);
   [..]
}

12.138.4.7 children()

int Fl_Table::children ( ) const [inline]
Returns the number of children in the table.
When using the Fl_Table as a container for FLTK widgets, this method returns how many child widgets the table has.
See also

    child(int)

12.138.4.8 clear()

virtual void Fl_Table::clear ( ) [inline], [virtual]
Clears the table to zero rows (rows(0)), zero columns (cols(0)), and clears any widgets (table->clear()) that were added with begin()/end() or add()/insert()/etc.
See also

    rows(int), cols(int)

Reimplemented in Fl_Table_Row.

12.138.4.9 col_header()

void Fl_Table::col_header ( int flag ) [inline]
Enable or disable column headers.
If changed, the table is redrawn.

12.138.4.10 col_resize()

void Fl_Table::col_resize ( int flag ) [inline]
Allows/disallows column resizing by the user.
1=allow interactive resizing, 0=disallow interactive resizing. Since interactive resizing is done via the column headers, col_header() must also be enabled to allow resizing.

12.138.4.11 col_resize_min()

void Fl_Table::col_resize_min ( int val ) [inline]
Sets the current column minimum resize value.
This is used to prevent the user from interactively resizing any column to be smaller than 'pixels'. Must be a value >=1.

12.138.4.12 col_width()

void Fl_Table::col_width ( int col, int width )
Sets the width of the specified column in pixels, and the table is redrawn.
callback() will be invoked with CONTEXT_RC_RESIZE if the column's width was actually changed, and when() is FL_WHEN_CHANGED.
12.138.4.13  col_width_all()  

    void Fl_Table::col_width_all ( 
    int width) [inline]

Convenience method to set the width of all columns to the same value, in pixels. The screen is redrawn.

12.138.4.14  cursor2rowcol()  

    Fl_Table::TableContext Fl_Table::cursor2rowcol ( 
    int & R, 
    int & C, 
    ResizeFlag & resizeflag) [protected]

Find row/col for the recent mouse event. Returns the context, and the row/column values in R/C. Also returns 'resizeflag' if mouse is hovered over a resize boundary.

12.138.4.15  damage_zone()  

    void Fl_Table::damage_zone ( 
    int r1, 
    int c1, 
    int r2, 
    int c2, 
    int r3 = 0, 
    int c3 = 0 ) [protected]

Sets the damage zone to the specified row/col values. Calls redraw_range().

12.138.4.16  do_callback()  

    void Fl_Table::do_callback ( 
    TableContext context, 
    int row, 
    int col) [inline]

Calls the widget callback. Saves the specified 'context', 'row', and 'col' values, so that the user's callback can then access them with the member functions callback_context(), callback_row() and callback_col().

12.138.4.17  draw()  

    void Fl_Table::draw ( 
    void) [protected], [virtual]

Draws the entire Fl_Table. Lets fltk widgets draw themselves first, followed by the cells via calls to draw_cell(). Reimplemented from Fl_Group.

12.138.4.18  draw_cell()  

    virtual void Fl_Table::draw_cell ( 
    TableContext context, 
    int R = 0, 
    int C = 0, 
    int X = 0, 
    int Y = 0, 
    int W = 0, 
    int H = 0 ) [inline], [protected], [virtual]

Subclass should override this method to handle drawing the cells. This method will be called whenever the table is redrawn, once per cell.
Only cells that are completely (or partially) visible will be told to draw. `context` will be one of the following:

| Fl_Table::CONTEXT_STARTPAGE | When table, or parts of the table, are about to be redrawn. Use to initialize static data, such as font selections. R/C will be zero, X/Y/W/H will be the dimensions of the table's entire data area. (Useful for locking a database before accessing; see also `visible_cells()`) |
| Fl_Table::CONTEXT_ENDPAGE | When table has completed being redrawn. R/C will be zero, X/Y/W/H dimensions of table's data area. (Useful for unlocking a database after accessing) |
| Fl_Table::CONTEXT_ROW_HEADER | Whenever a row header cell needs to be drawn. R will be the row number of the header being redrawn, C will be zero, X/Y/W/H will be the fltk drawing area of the row header in the window |
| Fl_Table::CONTEXT_COL_HEADER | Whenever a column header cell needs to be drawn. R will be zero, C will be the column number of the header being redrawn, X/Y/W/H will be the fltk drawing area of the column header in the window |
| Fl_Table::CONTEXT_CELL | Whenever a data cell in the table needs to be drawn. R/C will be the row/column of the cell to be drawn, X/Y/W/H will be the fltk drawing area of the cell in the window |
| Fl_Table::CONTEXT_RC_RESIZE | Whenever table or row/column is resized or scrolled, either interactively or via `col_width()` or `row_height()`. R/C/X/Y/W/H will all be zero. Useful for fltk containers that need to resize or move the child fltk widgets. |

R and C will be set to the row and column number of the cell being drawn. In the case of row headers, C will be 0. In the case of column headers, R will be 0. X/Y/W/H will be the position and dimensions of where the cell should be drawn.

In the case of custom widgets, a minimal `draw_cell()` override might look like the following. With custom widgets it is up to the caller to handle drawing everything within the dimensions of the cell, including handling the selection color. Note all clipping must be handled as well; this allows drawing outside the dimensions of the cell if so desired for 'custom effects'.

```c
// This is called whenever Fl_Table wants you to draw a cell
void MyTable::draw_cell(TableContext context, int R=0, int C=0, int X=0, int Y=0, int W=0, int H=0) {
    static char s[40];
    sprintf(s, "%d/%d", R, C); // text for each cell
    switch (context) {
    case CONTEXT_STARTPAGE: // Fl_Table telling us it's starting to draw page
        fl_font(Fl_HELVETICA, 16);
        return;
    case CONTEXT_ROW_HEADER: // Fl_Table telling us to draw row/col headers
    case CONTEXT_COL_HEADER:
        fl_push_clip(X, Y, W, H);
        { fl_draw_box(Fl_THIN_UP_BOX, X, Y, W, H, color());
          fl_color(Fl_BLACK);
          fl_draw(s, X, Y, W, H, Fl_ALIGN_CENTER);
        }
        fl_pop_clip();
        return;
    case CONTEXT_CELL: // Fl_Table telling us to draw cells
        fl_push_clip(X, Y, W, H);
        { // BG COLOR
          fl_color(row_selected(R) ? selection_color() : Fl_WHITE);
          fl_rectf(X, Y, W, H);
        // TEXT
          fl_color(Fl_BLACK);
        }
    }
```
12.138 Fl_Table Class Reference

```cpp
fl_draw(s, X, Y, W, H, FL_ALIGN_CENTER); // BORDER
fl_color(FL_LIGHT2);
fl_rect(X, Y, W, H);
}
fl_pop_clip();
return;
default:
return;

//NOTREACHED
}

12.138.4.19 find_cell()

int Fl_Table::find_cell(TableContext context,
                       int R,
                       int C,
                       int & X,
                       int & Y,
                       int & W,
                       int & H) [protected]

Find a cell's X/Y/W/H region for the specified cell in row 'R', column 'C'.

Returns

- 0 – on success, XYWH returns the region of the specified cell.
- -1 – if R or C are out of range, and X/Y/W/H will be set to zero.

12.138.4.20 get_selection()

void Fl_Table::get_selection {
    int & row_top,
    int & col_left,
    int & row_bot,
    int & col_right
}

Gets the region of cells selected (highlighted).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_top</td>
<td>Returns the top row of selection area</td>
</tr>
<tr>
<td>col_left</td>
<td>Returns the left column of selection area</td>
</tr>
<tr>
<td>row_bot</td>
<td>Returns the bottom row of selection area</td>
</tr>
<tr>
<td>col_right</td>
<td>Returns the right column of selection area</td>
</tr>
</tbody>
</table>

12.138.4.21 handle()

int Fl_Table::handle (int e) [protected], [virtual]

Handle FLTK events.
Reimplemented from Fl_Group.
Reimplemented in Fl_Table_Row.

12.138.4.22 init_sizes()

void Fl_Table::init_sizes () [inline]

Resets the internal array of widget sizes and positions.
```
void Fl_Table::insert ( Fl_Widget & wgt, Fl_Widget * w2 ) [inline]

The specified widget is removed from its current group (if any) and inserted into Fl_Table’s group before widget ‘w2’. This will append if ‘w2’ is not in Fl_Table’s group.

int Fl_Table::is_interactive_resize ( ) [inline]

Returns 1 if someone is interactively resizing a row or column. You can currently call this only from within your callback().

int Fl_Table::is_selected ( int r, int c )

See if the cell at row r and column c is selected.

Returns

1 if the cell is selected, 0 if not.

int Fl_Table::move_cursor ( int R, int C, int shiftselect )

Moves the selection cursor a relative number of rows/columns specified by R/C. R/C can be positive or negative, depending on the direction to move. A value of 0 for R or C prevents cursor movement on that axis.
If shiftselect is set, the selection range is extended to the new cursor position. If clear, the cursor is simply moved, and any previous selection is cancelled.
Used mainly by keyboard events (e.g. Fl_Right, Fl_Home, Fl_End..) to let the user keyboard navigate the selection cursor around.
The scroll positions may be modified if the selection cursor traverses into cells off the screen’s edge.
Internal variables select_row/select_col and current_row/current_col are modified, among others.
Examples:
R=1, C=0 -- moves the selection cursor one row downward.
R=5, C=0 -- moves the selection cursor 5 rows downward.
R=-5, C=0 -- moves the cursor 5 rows upward.
R=2, C=2 -- moves the cursor 2 rows down and 2 columns to the right.

void Fl_Table::recalc_dimensions ( ) [protected]

Recalculate the dimensions of the table, and affect any children. Internally, Fl_Group::resize() and init_sizes() are called.

void Fl_Table::recalc_range ( int topRow, int botRow,
Define region of cells to be redrawn by specified range of rows/cols, and then sets damage(DAMAGE_CHILD). Extends any previously defined range to redraw.

12.138.4.29 resize()

void Fl_Table::resize (  
    int X,  
    int Y,  
    int W,  
    int H ) [virtual]

Handle resize events if user resizes parent window. This changes the size of Fl_Table, causing it to redraw. Reimplemented from Fl_Group.

12.138.4.30 row_col_clamp()

int Fl_Table::row_col_clamp (  
    TableContext context,  
    int & R,  
    int & C ) [protected]

Return specified row/col values R and C to within the table's current row/col limits.

Returns
0 if no changes were made, or 1 if they were.

12.138.4.31 row_header()

void Fl_Table::row_header (  
    int flag ) [inline]

Enables/disables showing the row headers.  
1=enabled, 0=disabled. If changed, the table is redrawn.

12.138.4.32 row_height()

void Fl_Table::row_height (  
    int row,  
    int height )

Sets the height of the specified row in pixels, and the table is redrawn.  
callback() will be invoked with CONTEXT_RC_RESIZE if the row's height was actually changed, and when() is FL_WHEN_CHANGED.

12.138.4.33 row_height_all()

void Fl_Table::row_height_all (  
    int height ) [inline]

Convenience method to set the height of all rows to the same value, in pixels. The screen is redrawn.

12.138.4.34 row_resize()

void Fl_Table::row_resize (  
    int flag ) [inline]

Allows/disallows row resizing by the user.  
1=allow interactive resizing, 0=disallow interactive resizing. Since interactive resizing is done via the row headers, row_header() must also be enabled to allow resizing.
12.138.4.35  row_resize_min()

void Fl_Table::row_resize_min (  
    int val  )  [inline]

Sets the current row minimum resize value.  
This is used to prevent the user from interactively resizing any row to be smaller than 'pixels'.  Must be a value >=1.

12.138.4.36  rows()

void Fl_Table::rows (  
    int val  )  [virtual]

Sets the number of rows in the table, and the table is redrawn.  
Reimplemented in Fl_Table_Row.

12.138.4.37  scrollbar_size() [1/2]

int Fl_Table::scrollbar_size ( ) const  [inline]

Gets the current size of the scrollbars' troughs, in pixels.  
If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.  

Returns  
Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also  
Fl::scrollbar_size(int)

12.138.4.38  scrollbar_size() [2/2]

void Fl_Table::scrollbar_size (  
    int newSize  )  [inline]

Sets the pixel size of the scrollbars' troughs to newSize, in pixels.  
Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars.  This ensures your application has a consistent UI, is the default behavior, and is normally what you want.  
Only use THIS method if you really need to override the global scrollbar size.  The need for this should be rare.  
Setting newSize to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.  

Parameters  

| in     | newSize | Sets the scrollbar size in pixels.  
|--------|---------| If 0 (default), scrollbar size tracks the global Fl::scrollbar_size() |

See also  
Fl::scrollbar_size()

12.138.4.39  set_selection()

void Fl_Table::set_selection (  
    int row_top,  
    int col_left,  
    int row_bot,  
    int col_right )

Sets the region of cells to be selected (highlighted).
So for instance, set_selection(0,0,0,0) selects the top/left cell in the table. And set_selection(0,0,1,1) selects the four cells in rows 0 and 1, column 0 and 1.
To deselect all cells, use set_selection(-1,-1,-1,-1);

Parameters

| in | row_top | Top row of selection area |
| in | col_left | Left column of selection area |
| in | row_bot | Bottom row of selection area |
| in | col_right | Right column of selection area |

12.138.4.40 tab_cell_nav() [1/2]

int Fl_Table::tab_cell_nav ( ) const [inline]
Get state of table's 'Tab' key cell navigation flag.

Returns

1 if Tab configured to navigate cells in table
0 to navigate widget focus (default)

See also

tab_cell_nav(int)

12.138.4.41 tab_cell_nav() [2/2]

void Fl_Table::tab_cell_nav ( int val ) [inline]
Flag to control if Tab navigates table cells or not.
If on, Tab key navigates table cells. If off, Tab key navigates fltk widget focus. (default)
As of fltk 1.3, the default behavior of the Tab key is to navigate focus off of the current widget, and on to the next one. But in some applications, it's useful for Tab to be used to navigate cells in the Fl_Table.

Parameters

| in | val | If val is 1, Tab key navigates cells in table, not fltk widgets.
|    |     | If val is 0, Tab key will advance focus to the next fltk widget (default), and does not navigate cells in table. |

12.138.4.42 table_box()

void Fl_Table::table_box ( Fl_Boxtype val ) [inline]
Sets the kind of box drawn around the data table, the default being FL_NO_BOX.
Changing this value will cause the table to redraw.

12.138.4.43 table_resized()

void Fl_Table::table_resized ( ) [protected]
Call this if table was resized, to recalculate internal data.
Calls recall_dimensions(), and recalculates scrollbar sizes.

12.138.4.44 table_scrolled()

void Fl_Table::table_scrolled ( ) [protected]
Recalculate internals after a scroll. Call this if table has been scrolled or resized. Does not handle `redraw()`. TODO: Assumes t[xywh] has already been recalculated.

### 12.138.4.45 `top_row()` [1/2]

```cpp
int Fl_Table::top_row ( ) [inline]
```

Returns the current top row shown in the table.
This row may be partially obscured.

### 12.138.4.46 `top_row()` [2/2]

```cpp
void Fl_Table::top_row ( int row ) [inline]
```

Sets which row should be at the top of the table, scrolling as necessary, and the table is redrawn.
If the table cannot be scrolled that far, it is scrolled as far as possible.

### 12.138.4.47 `visible_cells()`

```cpp
void Fl_Table::visible_cells ( int & r1, int & r2, int & c1, int & c2 ) [inline]
```

Returns the range of row and column numbers for all visible and partially visible cells in the table.
These values can be used e.g. by your `draw_cell()` routine during `CONTEXT_STARTPAGE` to figure out what cells
are about to be redrawn for the purposes of locking the data from a database before it's drawn.

```
leftcol rightcol
| toprow .. VISIBL-------------E|
| T A B L E | V I S I B L E |
| botrow .. '-------------|
```

E.g., in a table where the visible rows are 5-20, and the visible columns are 100-120, then those variables would be:

- `toprow = 5`
- `botrow = 20`
- `leftcol = 100`
- `rightcol = 120`

### 12.138.4.48 `when()`

```cpp
void Fl_Table::when ( Fl_When flags )
```

The `Fl_Widget::when()` function is used to set a group of flags, determining when the widget callback is called:

| FL_WHEN_CHANGED | callback() will be called when rows or columns are resized (interactively or via `col_width()` or `row_height()`), passing `CONTEXT_RC_RESIZE` via `callback_context()`.
| FL_WHEN_RELEASE | callback() will be called during `FL_RELEASE` events, such as when someone releases a mouse button somewhere on the table. |

The `callback()` routine is sent a `TableContext` that indicates the context the event occurred in, such as in a cell, in a header, or elsewhere on the table. When an event occurs in a cell or header, `callback_row()` and `callback_col()` can be used to determine the row and column. The callback can also look at the regular Fltk event values (i.e. `Fl::event()` and `Fl::event_button()`)) to determine what kind of event is occurring.

The documentation for this class was generated from the following files:
12.139 Fl_Table_Row Class Reference

A table with row selection capabilities.

```cpp
#include <Fl_Table_Row.H>
```

Inheritance diagram for Fl_Table_Row:

```
Fl_Widget
   |
   v
Fl_Group
   |
   v
Fl_Table
   |
   v
Fl_Table_Row
```

Public Types

- `enum TableRowSelectMode { SELECT_NONE, SELECT_SINGLE, SELECT_MULTI }`

Public Types inherited from Fl_Table

- `enum TableContext { CONTEXT_NONE = 0, CONTEXT_STARTPAGE = 0x01, CONTEXT_ENDPAGE = 0x02, CONTEXT_ROW_HEADER = 0x04, CONTEXT_COL_HEADER = 0x08, CONTEXT_CELL = 0x10, CONTEXT_TABLE = 0x20, CONTEXT_RC_RESIZE = 0x40 }
  The context bit flags for Fl_Table related callbacks.

Public Member Functions

- `void clear () FL_OVERRIDE`  
  Clears the table to zero rows (rows(0)), zero columns (cols(0)), and clears any widgets (table->clear()) that were added with begin()/end() or add()/insert()/etc.
- `Fl_Table_Row (int X, int Y, int W, int H, const char ∗=0)`
  The constructor for the Fl_Table_Row.
- `int row_selected (int row)`
  Checks to see if 'row' is selected.
- `int rows ()`
- `void rows (int val) FL_OVERRIDE`
  Sets the number of rows in the table, and the table is redrawn.
- `void select_all_rows (int flag=1)`  
  This convenience function changes the selection state for all rows based on 'flag'.
- `int select_row (int row, int flag=1)`  
  Changes the selection state for 'row', depending on the value of 'flag'.
- `TableRowSelectMode type () const`
- `void type (TableRowSelectMode val)`
  Sets the table selection mode.
- `~Fl_Table_Row ()`
  The destructor for the Fl_Table_Row.
Public Member Functions inherited from Fl_Table

- void add (Fl_Widget &wgt)
  The specified widget is removed from its current group (if any) and added to the end of Fl_Table's group.

- void add (Fl_Widget *wgt)
  The specified widget is removed from its current group (if any) and added to the end of Fl_Table's group.

- Fl_Widget *const *array ()
  Returns a pointer to the array of children.

- void begin ()

- void callback (Fl_Widget *, void *)
  Callbacks will be called depending on the setting of Fl_Widget::when().

- int callback_col ()
  Returns the current column the event occurred on.

- TableContext callback_context ()
  Returns the current 'table context'.

- int callback_row ()
  Returns the current row the event occurred on.

- Fl_Widget *child (int n) const
  Returns the child widget by an index.

- int children () const
  Returns the number of children in the table.

- int col_header ()
  Returns if column headers are enabled or not.

- void col_header (int flag)
  Enable or disable column headers.

- Fl_Color col_header_color ()
  Gets the color for column headers.

- void col_header_color (Fl_Color val)
  Sets the color for column headers and redraws the table.

- int col_header_height ()
  Gets the column header height.

- void col_header_height (int height)
  Sets the height in pixels for column headers and redraws the table.

- int col_position ()
  Returns the current column scroll position as a column number.

- void col_position (int col)
  Sets the horizontal scroll position so 'col' is at the left, and causes the screen to redraw.

- int col_resize ()
  Returns if column resizing by the user is allowed.

- void col_resize (int flag)
  Allows/disallows column resizing by the user.

- int col_resize_min ()
  Returns the current column minimum resize value.

- void col_resize_min (int val)
  Sets the current column minimum resize value.

- int col_width (int col)
  Returns the current width of the specified column in pixels.

- void col_width (int col, int width)
  Sets the width of the specified column in pixels, and the table is redrawn.

- void col_width_all (int width)
  Convenience method to set the width of all columns to the same value, in pixels.
• int **cols** ()
  
  Get the number of columns in the table.

• virtual void **cols** (int val)
  
  Set the number of columns in the table and redraw.

• void **do_callback** (TableContext context, int row, int col)
  
  Calls the widget callback.

• void **end** ()

• int **find** (const Fl_Widget &wgt) const
• int **find** (const Fl_Widget *wgt) const

  The constructor for Fl_Table.

• void **get_selection** (int &row_top, int &col_left, int &row_bot, int &col_right)

  Gets the region of cells selected (highlighted).

• void **init_sizes** ()

  Resets the internal array of widget sizes and positions.

• void **insert** (Fl_Widget &wgt, Fl_Widget *w2)
  
  The specified widget is removed from its current group (if any) and inserted into Fl_Table's group before widget 'w2'.

• void **insert** (Fl_Widget &wgt, int n)
  
  The specified widget is removed from its current group (if any) and inserted into the Fl_Table's group at position 'n'.

• int **is_interactive_resize** ()

  Returns 1 if someone is interactively resizing a row or column.

• int **is_selected** (int r, int c)

  See if the cell at row **r** and column **c** is selected.

• int **move_cursor** (int R, int C)

  Same as move_cursor(R,C,1);

• int **move_cursor** (int R, int C, int shiftselect)

  Moves the selection cursor a relative number of rows/columns specified by R/C.

• void **remove** (Fl_Widget &wgt)

  The specified widget is removed from Fl_Table's group.

• void **resize** (int X, int Y, int W, int H) FL_OVERRIDE

  Handle resize events if user resizes parent window.

• int **row_header** ()

  Returns if row headers are enabled or not.

• void **row_header** (int flag)

  Enables/disables showing the row headers.

• Fl_Color **row_header_color** ()

  Returns the current row header color.

• void **row_header_color** (Fl_Color val)

  Sets the row header color and causes the screen to redraw.

• int **row_header_width** ()

  Returns the current row header width (in pixels).

• void **row_header_width** (int width)

  Sets the row header width to n and causes the screen to redraw.

• int **row_height** (int row)

  Returns the current height of the specified row as a value in pixels.

• void **row_height** (int row, int height)

  Sets the height of the specified row in pixels, and the table is redrawn.

• void **row_height_all** (int height)

  Convenience method to set the height of all rows to the same value, in pixels.

• int **row_position** ()

  Returns the current row scroll position as a row number.
• void **row_position** (int row)
  Sets the vertical scroll position so 'row' is at the top, and causes the screen to redraw.

• int **row_resize** ()
  Returns if row resizing by the user is allowed.

• void **row_resize** (int flag)
  Allows/disallows row resizing by the user.

• int **row_resize_min** ()
  Returns the current row minimum resize value.

• void **row_resize_min** (int val)
  Sets the current row minimum resize value.

• int **rows** ()
  Returns the number of rows in the table.

• int **scrollbar_size** () const
  Gets the current size of the scrollbars' troughs, in pixels.

• void **scrollbar_size** (int newSize)
  Sets the pixel size of the scrollbars' troughs to newSize, in pixels.

• void **set_selection** (int row_top, int col_left, int row_bot, int col_right)
  Sets the region of cells to be selected (highlighted).

• int **tab_cell_nav** () const
  Get state of table's 'Tab' key cell navigation flag.

• void **tab_cell_nav** (int val)
  Flag to control if Tab navigates table cells or not.

• void **table_box** (Fl_Boxtype val)
  Sets the kind of box drawn around the data table, the default being FL_NO_BOX.

• Fl_Boxtype **table_box** (void)
  Returns the current box type used for the data table.

• int **top_row** ()
  Returns the current top row shown in the table.

• void **top_row** (int row)
  Sets which row should be at the top of the table, scrolling as necessary, and the table is redrawn.

• void **visible_cells** (int &r1, int &r2, int &c1, int &c2)
  Returns the range of row and column numbers for all visible and partially visible cells in the table.

• void **when** (Fl_When flags)
  The Fl_Widget::when() function is used to set a group of flags, determining when the widget callback is called:

• ~Fl_Table ()
  The destructor for Fl_Table.

Public Member Functions inherited from Fl_Group

• Fl_Widget * & _ddfdesign_kludge ()
  This is for forms compatibility only.

• void **add** (Fl_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.

• void **add** (Fl_Widget +o)
  See void Fl_Group::add(Fl_Widget &w)

• void **add_resizable** (Fl_Widget &o)
  Adds a widget to the group and makes it the resizable widget.

• Fl_Widget +const + array () const
  Returns a pointer to the array of children.

• Fl_Group const * as_group () const FL_OVERRIDE

• Fl_Group * as_group () FL_OVERRIDE
Returns an Fl_Group pointer if this widget is an Fl_Group.

- **void begin ()**
  
  Sets the current group so you can build the widget tree by just constructing the widgets.

- **Fl_Widget * child (int n) const**
  
  Returns array()[n].

- **int children () const**
  
  Returns how many child widgets the group has.

- **void clear ()**
  
  Deletes all child widgets from memory recursively.

- **unsigned int clip_children ()**
  
  Returns the current clipping mode.

- **void clip_children (int c)**
  
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- **virtual int delete_child (int n)**
  
  Removes the widget at index from the group and deletes it.

- **void end ()**
  
  Exactly the same as current(this->parent()).

- **int find (const Fl_Widget &o) const**
  
  See int Fl_Group::find(const Fl_Widget *w) const.

- **int find (const Fl_Widget *) const**
  
  Searches the child array for the widget and returns the index.

- **Fl_Group (int, int, int, int, const char *)=0**
  
  Creates a new Fl_Group widget using the given position, size, and label string.

- **void focus (Fl_Widget *)**
  
  This is for forms compatibility only.

- **void forms_end ()**
  
  Resets the internal array of widget sizes and positions.

- **void insert (Fl_Widget &, int i)**
  
  The widget is removed from its current group (if any) and then inserted into this group.

- **void insert (Fl_Widget &, Fl_Widget *)**
  
  This does insert(w, find(before)).

- **void remove (Fl_Widget &)**
  
  Removes a widget from the group but does not delete it.

- **void remove (Fl_Widget *)**
  
  Removes the widget o from the group.

- **void remove (int index)**
  
  Removes the widget at index from the group but does not delete it.

- **Fl_Widget * resizable () const**
  
  Returns the group's resizable widget.

- **void resizable (Fl_Widget &o)**
  
  Sets the group's resizable widget.

- **void resizable (Fl_Widget *)**
  
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- **virtual ~Fl_Group ()**
  
  The destructor also deletes all the children.
Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
    
    Activates the widget.
• unsigned int active () const
    
    Returns whether the widget is active.
• int active_r () const
    
    Returns whether the widget and all of its parents are active.
• Fl_Align align () const
    
    Gets the label alignment.
• void align (Fl_Align alignment)
    
    Sets the label alignment.
• long argument () const
    
    Gets the current user data (long) argument that is passed to the callback function.
• void argument (long v)
    
    Sets the current user data (long) argument that is passed to the callback function.
• virtual class Fl_Gl_Window ∗ as_gl_window ()
    
    Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const ∗ as_gl_window () const
• virtual Fl_Window ∗ as_window ()
    
    Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const ∗ as_window () const
• void bind_deimage (Fl_Image ∗ img)
    
    Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage (int f)
    
    Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image (Fl_Image ∗ img)
    
    Sets the image to use as part of the widget label when in the active state.
• void bind_image (int f)
    
    Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• Fl_Boxtype box () const
    
    Gets the box type of the widget.
• void box (Fl_Boxtype new_box)
    
    Sets the box type for the widget.
• Fl_Callback_p callback () const
    
    Gets the current callback function for the widget.
• void callback (Fl_Callback ∗ cb)
    
    Sets the current callback function for the widget.
• void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
    
    Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback ∗ cb, void ∗ p)
    
    Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 ∗ cb)
    
    Sets the current callback function for the widget.
• void callback (Fl_Callback1 ∗ cb, long p=0)
    
    Sets the current callback function for the widget.
• unsigned int changed () const
    
    Checks if the widget value changed since the last callback.
• void clear_active ()
Marks the widget as inactive without sending events or changing focus.

- **void clear_changed ()**  
  Marks the value of the widget as unchanged.

- **void clear_damage (uchar c=0)**  
  Clears or sets the damage flags.

- **void clear_output ()**  
  Sets a widget to accept input.

- **void clear_visible ()**  
  Hides the widget.

- **void clear_visible_focus ()**  
  Disables keyboard focus navigation with this widget.

- **Fl_Color color () const**  
  Gets the background color of the widget.

- **void color (Fl_Color bg)**  
  Sets the background color of the widget.

- **void color (Fl_Color bg, Fl_Color sel)**  
  Sets the background and selection color of the widget.

- **Fl_Color color2 () const**  
  For back compatibility only.

- **void color2 (unsigned a)**  
  For back compatibility only.

- **int contains (const Fl_Widget *w) const**  
  Checks if w is a child of this widget.

- **void copy_label (const char *new_label)**  
  Sets the current label.

- **void copy_tooltip (const char *text)**  
  Sets the current tooltip text.

- **uchar damage () const**  
  Returns non-zero if draw() needs to be called.

- **void damage (uchar c)**  
  Sets the damage bits for the widget.

- **void damage (uchar c, int x, int y, int w, int h)**  
  Sets the damage bits for an area inside the widget.

- **int damage_resize (int, int, int, int)**  
  Internal use only.

- **void deactivate ()**  
  Deactivates the widget.

- **Fl_Image *deimage ()**  
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image *deimage () const**  
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (Fl_Image &img)**  
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (Fl_Image *img)**  
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const**  
  Returns whether the inactive image is managed by the widget.

- **void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**  
  Calls the widget callback function with default arguments.

- **void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**  
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• FL_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.
• const FL_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (FL_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (FL_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (FL_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• FL_Color labelcolor () const
  Gets the label color.
• void labelcolor (FL_Color c)
  Sets the label color.
• FL_Font labelfont () const
  Gets the font to use.
• void labelfont (FL_Font f)
  Sets the font to use.
• FL_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (FL_Fontsize pix)
  Sets the font size in pixels.
• FL_Labeltype labeltype () const
  Gets the label type.
• void labeltype (FL_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
Returns if a widget is used for output only.

- **Fl_Group** - **parent () const**
  Returns a pointer to the parent widget.

- **void parent (Fl_Group *p)**
  Internal use only - "for hacks only".

- **void position (int X, int Y)**
  Repositions the window or widget.

- **void redraw ()**
  Schedules the drawing of the widget.

- **void redraw_label ()**
  Schedules the drawing of the label.

- **Fl_Color selection_color () const**
  Gets the selection color.

- **void selection_color (Fl_Color a)**
  Sets the selection color.

- **void set_active ()**
  Marks the widget as active without sending events or changing focus.

- **void set_changed ()**
  Marks the value of the widget as changed.

- **void set_output ()**
  Sets a widget to output only.

- **void set_visible ()**
  Makes the widget visible.

- **void set_visible_focus ()**
  Enables keyboard focus navigation with this widget.

- **int shortcut_label () const**
  Returns whether the widget's label uses '&' to indicate shortcuts.

- **void shortcut_label (int value)**
  Sets whether the widget's label uses '&' to indicate shortcuts.

- **virtual void show ()**
  Makes a widget visible.

- **void size (int W, int H)**
  Changes the size of the widget.

- **int take_focus ()**
  Gives the widget the keyboard focus.

- **unsigned int takesevents () const**
  Returns if the widget is able to take events.

- **int test_shortcut ()**
  Returns true if the widget's label contains the entered 'x' shortcut.

- **const char * tooltip () const**
  Gets the current tooltip text.

- **void tooltip (const char *text)**
  Sets the current tooltip text.

- **Fl_Window *top_window () const**
  Returns a pointer to the top-level window for the widget.

- **Fl_Window *top_window_offset (int &xoff, int &yoff) const**
  Finds the x/y offset of the current widget relative to the top-level window.

- **uchar type () const**
  Gets the widget type.

- **void type (uchar t)**
  Sets the widget type.
• int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void *user_data () const

  Gets the user data for this widget.

• void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.

• void **user_data** (void ∗v)

  Sets the user data for this widget.

• void **user_data** (void ∗v)

  Sets the user data for this widget.

• unsigned int **visible** () const

  Returns whether a widget is visible.

• unsigned int **visible_focus** () const

  Checks whether this widget has a visible focus.

• void **visible_focus** (int v)

  Modifies keyboard focus navigation.

• int **visible_r** () const

  Returns whether a widget and all its parents are visible.

• int w () const

  Gets the widget width.

• FL_When when () const

  Returns the conditions under which the callback is called.

• void when (uchar i)

  Sets the flags used to decide when a callback is called.

• Fl_Window *window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const

  Gets the widget position in its window.

• int y () const

  Gets the widget position in its window.

• virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

• int find_cell (TableContext context, int R, int C, int &X, int &Y, int &W, int &H)

• int handle (int event) FL_OVERRIDE

  Handle FLTK events.

Protected Member Functions inherited from Fl_Table

• void change_cursor (Fl_Cursor newcursor)

  Change mouse cursor to different type.

• long col_scroll_position (int col)

  Returns the scroll position (in pixels) of the specified column 'col'.

• TableContext cursor2rowcol (int &R, int &C, ResizeFlag &resizeflag)

  Find row/col for the recent mouse event.

• void damage_zone (int r1, int c1, int r2, int c2, int r3=0, int c3=0)

  Sets the damage zone to the specified row/col values.

• void draw () FL_OVERRIDE

  Draws the entire Fl_Table.

• virtual void draw_cell (TableContext context, int R=0, int C=0, int X=0, int Y=0, int W=0, int H=0)

  Subclass should override this method to handle drawing the cells.

• int find_cell (TableContext context, int R, int C, int &X, int &Y, int &W, int &H)
Find a cell's X/Y/W/H region for the specified cell in row 'R', column 'C'.

- **void** get_bounds (TableContext context, int &X, int &Y, int &W, int &H)
  Returns the (X,Y,W,H) bounding region for the specified 'context'.

- **int** is_fltk_container ()
  Does the table contain any child fltk widgets?

- **void** recalc_dimensions ()
  Recalculate the dimensions of the table, and affect any children.

- **void** redraw_range (int topRow, int botRow, int leftCol, int rightCol)
  Define region of cells to be redrawn by specified range of rows/cols, and then sets damage(DAMAGE_CHILD).

- **int** row_col_clamp (TableContext context, int &R, int &C)
  Return specified row/col values R and C to within the table's current row/col limits.

- **long** row_scroll_position (int row)
  Returns the scroll position (in pixels) of the specified 'row'.

- **void** table_resized ()
  Call this if table was resized, to recalculate internal data.

- **void** table_scrolled ()
  Recalculate internals after a scroll.

### Protected Member Functions inherited from Fl_Group

- **Fl_Rect** * bounds ()
  Returns the internal array of widget sizes and positions.

- **void** draw_child (Fl_Widget &widget) const
  Forces a child to redraw.

- **void** draw_children ()
  Draws all children of the group.

- **void** draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.

- **virtual int** on_insert (Fl_Widget ∗, int)
  Allow derived groups to act when a widget is added as a child.

- **virtual int** on_move (int, int)
  Allow derived groups to act when a widget is moved within the group.

- **virtual void** on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.

- **int** * sizes ()
  Returns the internal array of widget sizes and positions.

- **void** update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

### Protected Member Functions inherited from Fl_Widget

- **void** clear_flag (unsigned int c)
  Clears a flag in the flags mask.

- **void** draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- **void** draw_box () const
  Draws the widget box according its box style.

- **void** draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

- **void** draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- **void** draw_focus () const
Draws a focus rectangle around the widget.

• void `draw_focus` (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

• void `draw_focus` (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

• void `draw_label` () const
  Draws the widget's label at the defined label position.

• void `draw_label` (int, int, int, int) const
  Draws the label in an arbitrary bounding box.

• Fl_Widget (int x, int y, int w, int h, const char ∗ label=0L)
  Creates a widget at the given position and size.

• unsigned int `flags` () const
  Gets the widget flags mask.

• void `h` (int v)
  Internal use only.

• void `set_flag` (unsigned int c)
  Sets a flag in the flags mask.

• void `w` (int v)
  Internal use only.

• void `x` (int v)
  Internal use only.

• void `y` (int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

• static Fl_Group ∗ `current` ()
  Returns the currently active group.

• static void `current` (Fl_Group ∗ g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

• static void `default_callback` (Fl_Widget ∗ widget, void ∗ data)
  The default callback for all widgets that don't set a callback.

• static unsigned int `label_shortcut` (const char ∗ t)
  Returns the Unicode value of the '&x' shortcut in a given text.

• static int `test_shortcut` (const char ∗ t, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Table

• enum `ResizeFlag`
  
  | RESIZE_NONE = 0 | RESIZE_COL_LEFT = 1 | RESIZE_COL_RIGHT = 2 | RESIZE_ROW_ABOVE = 3 | RESIZE_ROW_BELOW = 4 |
Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOL TIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19, NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

Static Protected Member Functions inherited from Fl_Table

- static void scroll_cb (Fl_Widget *, void *)
  Callback for when someone moves a scrollbar.

Protected Attributes inherited from Fl_Table

- int botrow
  bottom row# of currently visible table on screen
- int current_col
  selection cursor's current column (-1 if none)
- int current_row
  selection cursor's current row (-1 if none)
- Fl_Scrollbar * hscrollbar
  child horizontal scrollbar widget
- int leftcol
  left column# of currently visible table on screen
- int leftcol_scrollpos
  precomputed scroll position for left column
- int rightcol
  right column# of currently visible table on screen
- int select_col
  extended selection column (-1 if none)
- int select_row
  extended selection row (-1 if none)
- Fl_Scroll * table
  child Fl_Scroll widget container for child Fltk widgets (if any)
- int table_h
  table's virtual height (in pixels)
- int table_w
  table's virtual width (in pixels)
- int tih
  Data table’s inner h dimension, inside bounding box. See Table Dimension Diagram.
- int tiw
  Data table’s inner w dimension, inside bounding box. See Table Dimension Diagram.
- int tix
  Data table’s inner x dimension, inside bounding box. See Table Dimension Diagram.
• int \texttt{tiy}
  Data table's inner \texttt{y} dimension, inside bounding box. See Table Dimension Diagram.

• int \texttt{toh}
  Data table's outer \texttt{h} dimension, outside bounding box. See Table Dimension Diagram.

• int \texttt{toprow}
  \texttt{top row\#} of currently visible table on screen

• int \texttt{toprow\_scrollpos}
  precomputed scroll position for top row

• int \texttt{tow}
  Data table's outer \texttt{w} dimension, outside bounding box. See Table Dimension Diagram.

• int \texttt{tox}
  Data table's outer \texttt{x} dimension, outside bounding box. See Table Dimension Diagram.

• int \texttt{toy}
  Data table's outer \texttt{y} dimension, outside bounding box. See Table Dimension Diagram.

• \texttt{Fl\_Scrollbar\* vscrollbar}
  child vertical scrollbar widget

• int \texttt{wih}
  Table widget's inner \texttt{h} dimension, inside bounding box. See Table Dimension Diagram.

• int \texttt{wiw}
  Table widget's inner \texttt{w} dimension, inside bounding box. See Table Dimension Diagram.

• int \texttt{wix}
  Table widget's inner \texttt{x} dimension, inside bounding box. See Table Dimension Diagram.

• int \texttt{wiy}
  Table widget's inner \texttt{y} dimension, inside bounding box. See Table Dimension Diagram.

12.139.1 Detailed Description

A table with row selection capabilities.

This class implements a simple table with the ability to select rows. This widget is similar to an \texttt{Fl\_Browser} with columns. Most methods of importance will be found in the \texttt{Fl\_Table} widget, such as \texttt{Fl\_Table::rows()} and \texttt{Fl\_Table::cols()}. To be useful it must be subclassed and at minimum the \texttt{draw\_cell()} method must be overridden to provide the content of the cells. This widget does \texttt{not} manage the cell's data content; it is up to the parent class's \texttt{draw\_cell()} method override to provide this.

Events on the cells and/or headings generate callbacks when they are clicked by the user. You control when events are generated based on the values you supply for \texttt{Fl\_Table::when()}.

12.139.2 Constructor & Destructor Documentation

12.139.2.1 \texttt{Fl\_Table\_Row()}\newline
\texttt{Fl\_Table\_Row::Fl\_Table\_Row (} \newline
  \texttt{int X,} \newline
  \texttt{int Y,} \newline
  \texttt{int W,} \newline
  \texttt{int H,} \newline
  \texttt{const char \* l = 0 } \texttt{)} \texttt{[inline]}

The constructor for the \texttt{Fl\_Table\_Row}.\newline
This creates an empty table with no rows or columns, with headers and row/column resize behavior disabled.

12.139.2.2 \texttt{~Fl\_Table\_Row()}\newline
\texttt{Fl\_Table\_Row::~Fl\_Table\_Row ( } \texttt{)[inline]}

The destructor for the \texttt{Fl\_Table\_Row}.\newline
Destroyes the table and its associated widgets.
12.139.3 Member Function Documentation

12.139.3.1 clear()

```cpp
void Fl_Table_Row::clear() [inline], [virtual]
```
Clears the table to zero rows (rows(0)), zero columns (cols(0)), and clears any widgets (table->clear()) that were added with begin()//end() or add()//insert()//etc.

See also

- rows(int), cols(int)

Reimplemented from Fl_Table.

12.139.3.2 handle()

```cpp
int Fl_Table_Row::handle(int e) [protected], [virtual]
```
Handle FLTK events.
Reimplemented from Fl_Table.

12.139.3.3 row_selected()

```cpp
int Fl_Table_Row::row_selected(int row)
```
Checks to see if ‘row’ is selected.
Returns 1 if selected, 0 if not. You can change the selection of a row by clicking on it, or by using select_row(row, flag)

12.139.3.4 rows()

```cpp
void Fl_Table_Row::rows(int val) [virtual]
```
Sets the number of rows in the table, and the table is redrawn.
Reimplemented from Fl_Table.

12.139.3.5 select_all_rows()

```cpp
void Fl_Table_Row::select_all_rows(int flag = 1)
```
This convenience function changes the selection state for all rows based on ‘flag’. 0=deselect, 1=select, 2=toggle existing state.

12.139.3.6 select_row()

```cpp
int Fl_Table_Row::select_row(int row, int flag = 1)
```
Changes the selection state for ‘row’, depending on the value of ‘flag’. 0=deselected, 1=select, 2=toggle existing state.

12.139.3.7 type()

```cpp
void Fl_Table_Row::type(TableRowSelectMode val)
```
Sets the table selection mode.

- Fl_Table_Row::SELECT_NONE - No selection allowed
- Fl_Table_Row::SELECT_SINGLE - Only single rows can be selected
- Fl_Table_Row::SELECT_MULTI - Multiple rows can be selected
The documentation for this class was generated from the following files:

- Fl_Table_Row.H
- Fl_Table_Row.cxx

# 12.140 Fl_Tabs Class Reference

The `Fl_Tabs` widget is a container widget that displays a set of tabs, with each tab representing a different child widget.

```c
#include <Fl_Tabs.H>
```

Inheritance diagram for `Fl_Tabs`:

```
+----+      +----+      +----+
|    |      |    |      |    |
| Fl_Widget | Fl_Group | Fl_Tabs |
```

## Public Types

- `enum { OVERFLOW_COMPRESS = 0, OVERFLOW_CLIP, OVERFLOW_PULLDOWN, OVERFLOW_DRAG }`

## Public Member Functions

- `void client_area (int &rx, int &ry, int &rw, int &rh, int tabh=0)` 
  Returns the position and size available to be used by its children.
- `Fl_Tabs (int X, int Y, int W, int H, const char *L=0)` 
  Creates a new `Fl_Tabs` widget using the given position, size, and label string.
- `int handle (int)` FL_OVERRIDE 
  Handle all events in the tabs area and forward the rest to the selected child.
- `void handle_overflow (int ov)` 
  Set a method to handle an overflowing tab bar.
- `Fl_Widget* push () const` 
  Returns the tab group for the tab the user has currently down-clicked on and remains over until FL_RELEASE.
- `int push (Fl_Widget *)` 
  This is called by the tab widget's `handle()` method to set the tab group widget the user last FL_PUSH'ed on.
- `Fl_Align tab_align () const` 
  Gets the tab label alignment.
- `void tab_align (Fl_Align a)` 
  Sets the tab label alignment.
- `Fl_Widget* value ()` 
  Gets the currently visible widget/tab.
- `int value (Fl_Widget *)` 
  Sets the widget to become the current visible widget/tab.
- `virtual Fl_Widget* which (int event_x, int event_y)` 
  Return a pointer to the child widget with a tab at the given coordinates.
- `virtual ~Fl_Tabs ()` 
  Delete allocated resources and destroy all children.
Public Member Functions inherited from Fl_Group

- Fl_Widget *&_ddfdesign_kludge ()
  This is for forms compatibility only.
- void add (Fl_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.
- void add (Fl_Widget *o)
  See void Fl_Group::add(Fl_Widget &w)
- void add_resizable (Fl_Widget &o)
  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const *array () const
  Returns a pointer to the array of children.
- Fl_Group const *as_group () const FL_OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()
  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget *child (int n) const
  Returns array()[n].
- int children () const
  Returns how many child widgets the group has.
- void clear ()
  Deletes all child widgets from memory recursively.
- unsigned int clip_children ()
  Returns the current clipping mode.
- void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.
- void end ()
  Exactly the same as current(this->parent()).
- int find (const Fl_Widget &o) const
  See int Fl_Group::find(const Fl_Widget *w) const.
- int find (const Fl_Widget *) const
  Searches the child array for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *)=0
  Creates a new Fl_Group widget using the given position, size, and label string.
- void focus (Fl_Widget *W)
- void forms_end ()
  This is for forms compatibility only.
- void init_sizes ()
  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)
  This does insert(w, find(before)).
- void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)
  Removes the widget o from the group.
- void remove (int index)
Removes the widget at index from the group but does not delete it.

- **Fl_Widget** * resizable () const
  Returns the group's resizable widget.

- **void resizable (Fl_Widget &o)**
  Sets the group's resizable widget.

- **void resizable (Fl_Widget *o)**
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- **virtual ~Fl_Group ()**
  The destructor also deletes all the children.

### Public Member Functions inherited from **Fl_Widget**

- **void _clear_fullscreen ()**
- **void _set_fullscreen ()**
- **void activate ()**
  Activates the widget.

- **unsigned int active () const**
  Returns whether the widget is active.

- **int active_r () const**
  Returns whether the widget and all of its parents are active.

- **Fl_Align align () const**
  Gets the label alignment.

- **void align (Fl_Align alignment)**
  Sets the label alignment.

- **long argument () const**
  Gets the current user data (long) argument that is passed to the callback function.

- **void argument (long v)**
  Sets the current user data (long) argument that is passed to the callback function.

- **virtual class Fl_Gl_Window * as_gl_window ()**
  Returns an **Fl_Gl_Window** pointer if this widget is an **Fl_Gl_Window**.

- **virtual class Fl_Gl_Window const * as_gl_window () const**

- **virtual Fl_Window * as_window ()**
  Returns an **Fl_Window** pointer if this widget is an **Fl_Window**.

- **virtual Fl_Window const * as_window () const**

- **void bind_deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the inactive state.

- **void bind_deimage (int f)**
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- **void bind_image (Fl_Image *img)**
  Sets the image to use as part of the widget label when in the active state.

- **void bind_image (int f)**
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype box () const**
  Gets the box type of the widget.

- **void box (Fl_Boxtype new_box)**
  Sets the box type for the widget.

- **Fl_Callback_p callback () const**
  Gets the current callback function for the widget.

- **void callback (Fl_Callback *cb)**
  Sets the current callback function for the widget.

- **void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)**
Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
  Checks if the widget value changed since the last callback.
- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()
  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)
  Clears or sets the damage flags.
- void clear_output ()
  Sets a widget to accept input.
- void clear_visible ()
  Hides the widget.
- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
- Fl_Color color () const
  Gets the background color of the widget.
- void color (Fl_Color bg)
  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
- Fl_Color color2 () const
  For back compatibility only.
- void color2 (unsigned a)
  For back compatibility only.
- int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.
- void copy_label (const char *new_label)
  Sets the current label.
- void copy_tooltip (const char *text)
  Sets the current tooltip text.
- uchar damage () const
  Returns non-zero if draw() needs to be called.
- void damage (uchar c)
  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
- int damage_resize (int, int, int, int)
  Internal use only.
- void deactivate ()
  Deactivates the widget.
- Fl_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
 Gets the label type.
• void labeltype ( Fl_Labeltype a )
  Sets the label type.
• void measure_label ( int &ww, int &hh ) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard ( bool needs )
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent ( Fl_Group *p )
  Internal use only - "for hacks only".
• void position ( int X, int Y )
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color ( Fl_Color a )
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label ( int value )
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size ( int W, int H )
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.
• const char * tooltip () const
  Gets the current tooltip text.
• **void tooltip(const char ∗text)**
  Sets the current tooltip text.

• **Fl_Window ∗ top_window() const**
  Returns a pointer to the top-level window for the widget.

• **Fl_Window ∗ top_window_offset(int &xoff, int &yoff) const**
  Finds the x/y offset of the current widget relative to the top-level window.

• **uchar type() const**
  Gets the widget type.

• **void type(uchar t)**
  Sets the widget type.

• **int use_accents_menu()**
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• **void ∗ user_data() const**
  Gets the user data for this widget.

• **void user_data(Fl_Callback_User_Data ∗v, bool auto_free)**
  Sets the user data for this widget.

• **void user_data(void ∗v)**
  Sets the user data for this widget.

• **unsigned int visible() const**
  Returns whether a widget is visible.

• **unsigned int visible_focus() const**
  Checks whether this widget has a visible focus.

• **void visible_focus(int v)**
  Modifies keyboard focus navigation.

• **int visible_r() const**
  Returns whether a widget and all its parents are visible.

• **int w() const**
  Gets the widget width.

• **Fl_When when() const**
  Returns the conditions under which the callback is called.

• **void when(uchar i)**
  Sets the flags used to decide when a callback is called.

• **Fl_Window ∗ window() const**
  Returns a pointer to the nearest parent window up the widget hierarchy.

• **int x() const**
  Gets the widget position in its window.

• **int y() const**
  Gets the widget position in its window.

• **virtual ~Fl_Widget()**
  Destroys the widget.

### Protected Member Functions

• **void check_overflow_menu()**
  Check if the tabs overflow and sets the has_overflow_menu flag accordingly.

• **virtual void clear_tab_positions()**
  Clear internal array of tab positions and widths.

• **void draw() FL_OVERRIDE**
  Draw the tabs area, the optional pulldown button, and all children.

• **void draw_overflow_menu_button()**
  Draw square button-like graphics with a down arrow in the top or bottom right corner.
• virtual void draw_tab (int x1, int x2, int W, int H, Fl_Widget *o, int flags, int sel)
  Draw a tab in the top or bottom tabs area.
• void handle_overflow_menu ()
  This is called when the user clicks the overflow pulldown menu button.
• virtual int hit_close (Fl_Widget *o, int event_x, int event_y)
  Check whether the coordinates fall within the "close" button area of the tab.
• virtual int hit_overflow_menu (int event_x, int event_y)
  Determine if the coordinates are in the area of the overflow menu button.
• virtual int hit_tabs_area (int event_x, int event_y)
  Determine if the coordinates are within the tabs area.
• int on_insert (Fl_Widget *, int) FL_OVERRIDE
  Make sure that we redraw all tabs when new children are added.
• int on_move (int, int) FL_OVERRIDE
  Make sure that we redraw all tabs when children are moved.
• void on_remove (int) FL_OVERRIDE
  Make sure that we redraw all tabs when new children are removed.
• virtual void redraw_tabs ()
  Redraw all tabs (and only the tabs).
• void resize (int, int, int, int) FL_OVERRIDE
  Make sure that we redraw all tabs when the widget size changes.
• virtual int tab_height ()
  Return space (height) in pixels usable for tabs.
• virtual int tab_positions ()
  Calculate tab positions and widths.

Protected Member Functions inherited from Fl_Group
• Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
• void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.
• void draw_children ()
  Draws all children of the group.
• void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.
• int * sizes ()
  Returns the internal array of widget sizes and positions.
• void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget
• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const
  Draws the widget box according its box style.
• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
Draws a box of type t, of color c at the position X,Y and size W,H.

- void `draw_focus ()` const
  Draws a focus rectangle around the widget.

- void `draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const`
  Draws a focus rectangle around the widget.

- void `draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const`
  Draws a focus box for the widget at the given position and size.

- void `draw_label ()` const
  Draws the widget's label at the defined label position.

- void `draw_label (int, int, int, int) const`
  Draws the label in an arbitrary bounding box.

- `Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)`
  Creates a widget at the given position and size.

- unsigned int `flags ()` const
  Gets the widget flags mask.

- void `h (int v)`
  Internal use only.

- void `set_flag (unsigned int c)`
  Sets a flag in the flags mask.

- void `w (int v)`
  Internal use only.

- void `x (int v)`
  Internal use only.

- void `y (int v)`
  Internal use only.

Protected Attributes

- int `has_overflow_menu`
  Set in OVERFLOW_PULLDOWN mode if tabs overflow. The actual menu array is created only on demand

- int `overflow_type`

- `Fl_Align tab_align_`
  Tab label alignment

- int `tab_count`
  Array size of tab positions etc.

- int ∗ `tab_flags`
  Array of tab flag of tabs per child.

- int `tab_offset`
  For pulldown and drag overflow, this is the horizontal offset when the tabs bar is dragged by the user

- int ∗ `tab_pos`
  Array of x-offsets of tabs per child + 1.

- int ∗ `tab_width`
  Array of widths of tabs per child.

Additional Inherited Members

Static Public Member Functions inherited from `Fl_Group`

- static `Fl_Group ∗ current ()`
  Returns the currently active group.

- static void `current (Fl_Group ∗g)`
  Sets the current group.
Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  
  The default callback for all widgets that don’t set a callback.
- static unsigned int label_shortcut (const char *t)
  
  Returns the Unicode value of the ‘&x’ shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)
  
  Returns true if the given text `t` contains the entered ‘&x’ shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , Copied_Label = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , Copied_Tooltip = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }

  flags possible values enumeration.

12.140.1 Detailed Description

The Fl_Tabs widget is a container widget that displays a set of tabs, with each tab representing a different child widget.

The user can select a tab by clicking on it, and the corresponding child widget will be displayed. The Fl_Tabs widget is useful for organizing a large number of controls or other widgets into a compact space, allowing the user to switch between different sets of controls as needed.

![Fl_Tabs](image)

Clicking the tab makes a child visible() by calling show() on it, and all other children are made invisible by calling hide() on them. Usually the children are Fl_Group widgets containing several widgets themselves.

Each child makes a card, and its label() is printed on the card tab, including the label font and style. The selection color of that child is used to color the tab, while the color of the child determines the background color of the pane. ‘&’ in labels are used to prefix a shortcut that is drawn underlined and that activates the corresponding tab; repeated ‘&’ avoids that.

The size of the tabs is controlled by the bounding box of the children (there should be some space between the children and the edge of the Fl_Tabs), and the tabs may be placed "inverted" on the bottom - this is determined by
which gap is larger. It is easiest to lay this out in FLUID, using the FLUID browser to select each child group and
resize them until the tabs look the way you want them to.
The background area behind and to the right of the tabs is "transparent", exposing the background detail of the
parent. The value of Fl_Tabs::box() does not affect this area. So if Fl_Tabs is resized by itself without the parent,
force the appropriate parent (visible behind the tabs) to redraw() to prevent artifacts.
See "Resizing Caveats" below on how to keep tab heights constant. See "Callback's Use Of when()" on how to
control the details of how clicks invoke the callback().
A typical use of the Fl_Tabs widget:

```cpp
// Typical use of Fl_Tabs
Fl_Tabs *tabs = new Fl_Tabs(10,10,300,200);
{
    Fl_Group *grp1 = new Fl_Group(20,30,280,170,"Tab1");
    { ..widgets that go in tab1.. }
    grp1->end();
    Fl_Group *grp2 = new Fl_Group(20,30,280,170,"Tab2");
    { ..widgets that go in tab2.. }
    grp2->end();
} tabs->end();
```

Default Appearance
The appearance of each "tab" is taken from the label() and color() of the child group corresponding to that "tab" and
panel. Where the "tabs" appear depends on the position and size of the child groups that make up the panels within
the Fl_Tabs widget, i.e. whether there is more space above or below them. The height of the "tabs" depends on
how much free space is available.

![Figure 12.51 Fl_Tabs Default Appearance](image1)

Highlighting The Selected Tab
The selected "tab" can be highlighted further by setting the selection_color() of the Fl_Tab itself, e.g.

```cpp
..
tabs = new Fl_Tabs(..);
tabs->selection_color(FL_DARK3);
..```

The result of the above looks like:

![Figure 12.52 Highlighting the selected tab](image2)

Uniform Tab and Panel Appearance
In order to have uniform tab and panel appearance, not only must the color() and selection_color() for each child
group be set, but also the selection_color() of the Fl_Tab itself any time a new "tab" is selected. This can be achieved
within the Fl_Tab callback, e.g.
void MyTabCallback(Fl_Widget *w, void*) {
    Fl_Tabs *tabs = (Fl_Tabs*)w;
    // When tab changed, make sure it has same color as its group
    tabs->selection_color( (tabs->value())->color() );
}

int main(..) {
    // Define tabs widget
    tabs = new Fl_Tabs(...);
    tabs->callback(MyTabCallback);

    // Create three tabs each colored differently
    grp1 = new Fl_Group(.. "One");
    grp1->color(9);
    grp1->selection_color(9);
    grp1->end();

    grp2 = new Fl_Group(.. "Two");
    grp2->color(10);
    grp2->selection_color(10);
    grp2->end();

    grp3 = new Fl_Group(.. "Three");
    grp3->color(14);
    grp3->selection_color(14);
    grp3->end();

    // Make sure default tab has same color as its group
    tabs->selection_color( (tab->value())->color() );

    return Fl::run();
}

The result of the above looks like:

![Fl Tabs with uniform colors](image)

If Fl_Tabs has no children, the widget will be drawn as a flat rectangle in the background color set by color().

**Close Button on Tabs**

The Fl_Tabs widget allows you to specify that a child widget should display a close button in its tab. If the FL_WHEN_CLOSED flag is set for the child widget, an "X" symbol will be displayed to the left of the label text in the tab. When the close button is clicked, the child widget's callback function will be called with the FL_REASON_CLOSED reason. It is then the responsibility of the child widget to remove itself from the Fl_Tabs container.

Tabs that are in a compressed state will not display a close button until they are fully expanded.

**Overflowing Tabs**

When the combined width of the tabs exceeds that of the Fl_Tabs widget, the tabs will overflow. Fl_Tabs provides four options for managing tabs overflow:

- **Fl_Tabs::OVERFLOW_COMPRESS**: proportionally compress the tabs to the left and right of the selected tab until they all fit within the widget.
- **Fl_Tabs::OVERFLOW_CLIP**: clips any tabs that extend beyond the right edge of the Fl_Tabs widget, making some tabs unreachable.
- **Fl_Tabs::OVERFLOW_PULLDOWN**: doesn't compress the tabs but instead generates a pulldown menu at the right end of the tabs area, displaying all available tabs.
- **Fl_Tabs::OVERFLOW_DRAG**: maintains the tabs' original sizes, allowing horizontal dragging of the tabs area using the mouse, a horizontal mouse wheel, or the horizontal scrolling gesture on touchpads.

**Resizing Caveats**

When Fl_Tabs is resized vertically, the default behavior scales the tab's height as well as its children. To keep the tab height constant during resizing, set the tab widget's resizable() to one of the tab's child groups, i.e.
Callback's Use Of when()

As of FLTK 1.3.3, Fl_Tabs() supports the following flags for when():

- **FL_WHEN_NEVER** – callback never invoked (all flags off)
- **FL_WHEN_CHANGED** – if flag set, invokes callback when a tab has been changed (on click or keyboard navigation)
- **FL_WHEN_NOT_CHANGED** – if flag set, invokes callback when the tabs remain unchanged (on click or keyboard navigation)
- **FL_WHEN_RELEASE** – if flag set, invokes callback on RELEASE of mouse button or keyboard navigation

Notes:

1. The above flags can be logically OR-ed (|) or added (+) to combine behaviors.
2. The default value for when() is FL_WHEN_RELEASE (inherited from Fl_Widget).
3. If FL_WHEN_RELEASE is the only flag specified, the behavior will be as if (FL_WHEN_RELEASE|FL_WHEN_CHANGED) was specified.
4. The value of changed() will be valid during the callback.
5. If both FL_WHEN_CHANGED and FL_WHEN_NOT_CHANGED are specified, the callback is invoked whether the tab has been changed or not. The changed() method can be used to determine the cause.
6. FL_WHEN_NOT_CHANGED can happen if someone clicks on an already selected tab, or if a keyboard navigation attempt results in no change to the tabs, such as using the arrow keys while at the left or right end of the tabs.
7. Fl::callback_reason() returns FL_REASON_SELECTED or FL_REASON_RESELECTED

**12.140.2 Member Enumeration Documentation**

**12.140.2.1 anonymous enum**

anonymous enum

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVERFLOW_COMPRESS</td>
<td>Tabs will be compressed and overlaid on top of each other.</td>
</tr>
<tr>
<td>OVERFLOW_CLIP</td>
<td>Only the first tabs that fit will be displayed.</td>
</tr>
<tr>
<td>OVERFLOW_PULLDOWN</td>
<td>Tabs that do not fit will be placed in a pull-down menu.</td>
</tr>
</tbody>
</table>
| OVERFLOW_DRAG       | The tab bar can be dragged horizontally to reveal additional tabs.

**12.140.3 Constructor & Destructor Documentation**

**12.140.3.1 Fl_Tabs()**

Fl_Tabs::Fl_Tabs {
int X,
int Y,
int W,
int H,
const char ∗ L = 0 )

Creates a new Fl_Tabs widget using the given position, size, and label string. The default boxtype is FL_THIN_UP_BOX.

Use add(Fl_Widget∗) to add each child, which are usually Fl_Group widgets. The children should be sized to stay away from the top or bottom edge of the Fl_Tabs widget, which is where the tabs will be drawn. All children of Fl_Tabs should have the same size and exactly fit on top of each other. They should only leave space above or below where the tabs will go, but not on the sides. If the first child of Fl_Tabs is set to "resizable()")", the riders will not resize when the tabs are resized.

The destructor also deletes all the children. This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the Fl_Tabs and all of its children can be automatic (local) variables, but you must declare the Fl_Tabs widget first so that it is destroyed last.

12.140.4 Member Function Documentation

12.140.4.1 clear_tab_positions()

void Fl_Tabs::clear_tab_positions ( ) [protected], [virtual]
Clear internal array of tab positions and widths.

See also

tab_positions().

12.140.4.2 client_area()

void Fl_Tabs::client_area ( 
    int & rx,
    int & ry,
    int & rw,
    int & rh,
    int tabh = 0 )

Returns the position and size available to be used by its children. If there isn't any child yet the tabh parameter will be used to calculate the return values. This assumes that the children's labelsize is the same as the Fl_Tabs' labelsize and adds a small border. If there are already children, the values of child(0) are returned, and tabh is ignored.

Note

Children should always use the same positions and sizes.

tabh can be one of

- 0: calculate label size, tabs on top
- -1: calculate label size, tabs on bottom
- > 0: use given tabh value, tabs on top (height = tabh)
- < -1: use given tabh value, tabs on bottom (height = -tabh)

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tabh</th>
<th>position and optional height of tabs (see above)</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>rx,ry,rw, rh</td>
<td>(x,y,w,h) of client area for children</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Since FLTK 1.3.0

12.140.4.3 draw()

```cpp
void Fl_Tabs::draw () [protected], [virtual]
```

Draw the tabs area, the optional pulldown button, and all children.
Reimplemented from Fl_Group.

12.140.4.4 draw_tab()

```cpp
void Fl_Tabs::draw_tab ( int x1,
int x2,
int W,
int H,
Fl_Widget * o,
int flags,
int what ) [protected], [virtual]
```

Draw a tab in the top or bottom tabs area.
Tabs can be selected, or on the left or right side of the selected tab. If overlapping, left tabs are drawn bottom to top using clipping. The selected tab is then the topmost, followed by the right side tabs drawn top to bottom.
Tabs with the FL_WHEN_CLOSE bit set will draw a cross on their left side only if they are not compressed/overlapping.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>x1</th>
<th>horizontal position of the left visible edge of the tab</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x2</td>
<td>horizontal position of the following tab</td>
</tr>
<tr>
<td></td>
<td>W,H</td>
<td>width and height of the tab</td>
</tr>
<tr>
<td></td>
<td>o</td>
<td>the child widget that corresponds to this tab</td>
</tr>
<tr>
<td></td>
<td>flags</td>
<td>if bit 1 is set, this tab is overlapped by another tab</td>
</tr>
<tr>
<td></td>
<td>what</td>
<td>can be LEFT, SELECTED, or RIGHT to indicate if the tab is to the left side or the right side of the selected tab, or the selected tab itself</td>
</tr>
</tbody>
</table>

12.140.4.5 handle()

```cpp
int Fl_Tabs::handle ( int event ) [virtual]
```

Handle all events in the tabs area and forward the rest to the selected child.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>event</th>
<th>handle this event</th>
</tr>
</thead>
</table>

**Returns**

1 if the event was handled

Reimplemented from Fl_Group.

12.140.4.6 handle_overflow()

```cpp
void Fl_Tabs::handle_overflow ( int ov )
```
Set a method to handle an overflowing tab bar.
The Fl_Tabs widget allows you to specify how to handle the situation where there are more tabs than can be displayed at once. The available options are:

- **OVERFLOW_COMPRESS**: Tabs will be compressed and overlaid on top of each other.
- **OVERFLOW_CLIP**: Only the first tabs that fit will be displayed.
- **OVERFLOW_PULLDOWN**: Tabs that do not fit will be placed in a pull-down menu.
- **OVERFLOW_DRAG**: The tab bar can be dragged horizontally to reveal additional tabs.

You can set the desired behavior using the `overflow()` method.

**Parameters**

- `overflow type`

**See also**

OVERFLOW_COMPRESS, OVERFLOW_CLIP, OVERFLOW_PULLDOWN, OVERFLOW_DRAG

12.140.4.7 handle_overflow_menu()

```cpp
void Fl_Tabs::handle_overflow_menu ( ) [protected]
```

This is called when the user clicks the overflow pulldown menu button.

This method creates a menu item array that contains the titles of all tabs in the Fl_Tabs group. Visible and invisible tabs are separated by dividers to indicate their state. The menu is then presented until the user selects an item or cancels. The chosen tab is then selected and made visible. The menu item array is the deleted.

12.140.4.8 hit_close()

```cpp
int Fl_Tabs::hit_close ( Fl_Widget * o, int event_x, int event_y ) [protected], [virtual]
```

Check whether the coordinates fall within the "close" button area of the tab.

The Fl_Tabs::hit_close() method checks whether the given event coordinates fall within the area of the "close" button on the tab of the specified child widget. This method should be called after the Fl_Tabs::which() method, which updates a lookup table used to determine the width of each tab.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>o</code></td>
<td>check the tab of this widget</td>
</tr>
<tr>
<td><code>event_x</code>, <code>event_y</code></td>
<td>event coordinates</td>
</tr>
</tbody>
</table>

**Returns**

1 if we hit the close button, and 0 otherwise

12.140.4.9 hit_overflow_menu()

```cpp
int Fl_Tabs::hit_overflow_menu ( int event_x, int event_y ) [protected], [virtual]
```
Determine if the coordinates are in the area of the overflow menu button.
Parameters

\begin{itemize}
\item \texttt{event\_x, event\_y} \rightarrow \text{event coordinates}
\end{itemize}

Returns

1 if we hit the overflow menu button, and 0 otherwise

\subsection{hit\_tabs\_area()}

\begin{verbatim}
int Fl_Tabs::hit_tabs_area (  
    int event\_x,  
    int event\_y ) [protected], [virtual]
\end{verbatim}

Determine if the coordinates are within the tabs area.

Parameters

\begin{itemize}
\item \texttt{event\_x, event\_y} \rightarrow \text{event coordinates}
\end{itemize}

Returns

1 if we hit the tabs area, and 0 otherwise

\subsection{on\_insert()}

\begin{verbatim}
int Fl_Tabs::on_insert (  
    Fl_Widget * candidate,  
    int index ) [protected], [virtual]
\end{verbatim}

Make sure that we redraw all tabs when new children are added.
Reimplemented from \texttt{Fl\_Group}.

\subsection{on\_move()}

\begin{verbatim}
int Fl_Tabs::on_move (  
    int a,  
    int b ) [protected], [virtual]
\end{verbatim}

Make sure that we redraw all tabs when children are moved.
Reimplemented from \texttt{Fl\_Group}.

\subsection{on\_remove()}

\begin{verbatim}
void Fl_Tabs::on_remove (  
    int index ) [protected], [virtual]
\end{verbatim}

Make sure that we redraw all tabs when new children are removed.
Reimplemented from \texttt{Fl\_Group}.

\subsection{push() [1/2]}

\begin{verbatim}
Fl_Widget * Fl_Tabs::push ( ) const [inline]
\end{verbatim}

Returns the tab group for the tab the user has currently down-clicked on and remains over until FL\_RELEASE. Otherwise, returns NULL.
While the user is down-clicked on a tab, the return value is the tab group for that tab. But as soon as the user releases, or drags off the tab with the button still down, the return value will be NULL.
See also

`push(Fl_Widget*)`.

### 12.140.4.15 push()

```cpp
int Fl_Tabs::push (Fl_Widget * o)
```

This is called by the tab widget's handle() method to set the tab group widget the user last FL_PUSH'ed on. Set back to zero on FL_RELEASE.

As of this writing, the value is mainly used by draw_tab() to determine whether or not to draw a 'down' box for the tab when it's clicked, and to turn it off if the user drags off it.

See also

`push()`.

### 12.140.4.16 redraw_tabs()

```cpp
void Fl_Tabs::redraw_tabs () [protected], [virtual]
```

Redraw all tabs (and only the tabs).

This method sets the Fl_Tab's damage flags so the tab area is redrawn.

### 12.140.4.17 resize()

```cpp
void Fl_Tabs::resize (int X, int Y, int W, int H) [protected], [virtual]
```

Make sure that we redraw all tabs when the widget size changes.

Reimplemented from Fl_Group.

### 12.140.4.18 tab_align()

```cpp
Fl_Align Fl_Tabs::tab_align () const [inline]
```

Gets the tab label alignment.

See also

`tab_align(Fl_Align)`

### 12.140.4.19 tab_align()

```cpp
void Fl_Tabs::tab_align (Fl_Align a) [inline]
```

Sets the tab label alignment.

The default is FL_ALIGN_CENTER so tab labels are centered, but since the label space is measured (per label) to fit the labels, there wouldn't be any difference if labels were aligned left or right.

If you want to show an image (icon) next to the group's label you can set a different label alignment. FL_ALIGN_IMAGE_NEXT_TO_TEXT is the recommended alignment to show the icon left of the text.

### 12.140.4.20 tab_height()

```cpp
int Fl_Tabs::tab_height () [protected], [virtual]
```

Return space (height) in pixels usable for tabs.

The calculated height is the largest space between all children and the upper and lower widget boundaries, respectively. If the space at the bottom is larger than at the top, the value will be negative and the tabs should be placed at the bottom.
Returns

Vertical space that can be used for the tabs.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>To put the tabs at the top of the widget.</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>To put the tabs on the bottom.</td>
</tr>
<tr>
<td>Full</td>
<td>Height, if <code>children() == 0</code>.</td>
</tr>
</tbody>
</table>

### 12.140.4.21 tab_positions()

```cpp
int Fl_Tabs::tab_positions ( ) [protected], [virtual]
```

Calculate tab positions and widths.

This protected method calculates the horizontal display positions and widths of all tabs. If the number of children `nc` (see below) is `> 0` three internal arrays are allocated, otherwise the arrays are free’d and the pointers are set to NULL. Note that the first array is larger (`nc+1`).

- `tab_pos[nc+1]` : The left edges of each tab plus a fake left edge for a tab past the right-hand one.
- `tab_width[nc]` : The width of each tab
- `tab_flags[nc]` : Flags, bit 0 is set if the tab is compressed

If needed, these arrays are (re)allocated. These positions are actually of the left edge of the slope. They are either separated by the correct distance or by `EXTRASPACE` or by zero.

In `OVERFLOW_COMPRESS` mode, tab positions and widths are compressed to make the entire tabs bar fit into the width of `Fl_Tabs` while keeping the selected tab fully visible.

In other overflow modes, the tabs area may be dragged horizontally using `tab_offset`. The `tab_pos` array is not adjusted to the horizontal offset, but starts at this-`x()` plus the box’s left margin.

The protected variable `tab_count` is set to the currently allocated size, i.e. the number of children (`nc`).

Returns

Index of the selected item

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>If the number of children is 0 (zero).</td>
</tr>
</tbody>
</table>

Note

Return values in 1.3 were not documented. Return values before Sep 2023 were documented as 1 based index and 0 if there were no children. This was actually never the case. It always returned a 0 based index and the (useless) value of also 0 if there were no children. The current version return -1 if there are no children.

For this method to work, only on single child should be selected. Calling the method `value()` before calling `tab_positions()` will ensure that exactly one child is selected and return a pointer to that child.

See also

`clear_tab_positions()`

### 12.140.4.22 value() [1/2]

```cpp
Fl_Widget * Fl_Tabs::value ( )
```

Gets the currently visible widget/tab.
The `Fl_Tabs::value()` method returns a pointer to the currently visible child widget of the `Fl_Tabs` container. The visible child is the first child that is currently being displayed, or the last child if none of the children are being displayed. If child widgets have been added, moved, or deleted, this method ensures that only one tab is visible at a time.

Returns

a pointer to the currently visible child

**12.140.4.23 value() [2/2]**

```cpp
int Fl_Tabs::value ( Fl_Widget * newvalue )
```

Sets the widget to become the current visible widget/tab.

The `Fl_Tabs::value()` method allows you to set a particular child widget of the `Fl_Tabs` container to be the currently visible widget. If the specified widget is a child of the `Fl_Tabs` container, it will be made visible and all other children will be hidden. The method returns 1 if the value was changed, and 0 if the specified value was already set.

Parameters

- **in newvalue** a pointer to a child widget

Returns

1 if a different tab was chosen

0 if there was no change (new value already set)

**12.140.4.24 which()**

```cpp
Fl_Widget * Fl_Tabs::which ( int event_x, int event_y ) [virtual]
```

Return a pointer to the child widget with a tab at the given coordinates.

The `Fl_Tabs::which()` method returns a pointer to the child widget of the `Fl_Tabs` container that corresponds to the tab at the given event coordinates. If the event coordinates are outside the area of the tabs or if the `Fl_Tabs` container has no children, the method returns NULL.

Parameters

- **event_x, event_y** event coordinates

Returns

pointer to the selected child widget, or NULL

**12.140.5 Member Data Documentation**

**12.140.5.1 overflow_type**

```cpp
int Fl_Tabs::overflow_type [protected]
```

See also

OVERFLOW_COMPRESS, OVERFLOW_CLIP, etc.

**12.140.5.2 tab_count**

```cpp
int Fl_Tabs::tab_count [protected]
```
Array size of tab positions etc.
See also
  
  tab_positions()

12.140.5.3  tab_flags

int* Fl_Tabs::tab_flags  [protected]
Array of tab flag of tabs per child.
See also
  
  tab_positions()

12.140.5.4  tab_pos

int* Fl_Tabs::tab_pos  [protected]
Array of x-offsets of tabs per child + 1.
See also
  
  tab_positions()

12.140.5.5  tab_width

int* Fl_Tabs::tab_width  [protected]
Array of widths of tabs per child.
See also
  
  tab_positions()

The documentation for this class was generated from the following files:
  
  • Fl_Tabs.H
  • Fl_Tabs.cxx

12.141  Fl_Terminal Class Reference

Terminal widget supporting Unicode/utf-8, ANSI/xterm escape codes with full RGB color control.
#include <Fl_Terminal.H>
Inheritance diagram for Fl_Terminal:

```
        Fl_Widget
         ↓
       Fl_Group
         ↓
       Fl_Terminal
```

Classes

- class CharStyle
- class Cursor
- class EscapeSeq
- class Margin
- class PartialUtf8Buf
- class RingBuffer
- class Selection
- class Utf8Char
Public Types

- enum Attrib {
  NORMAL = 0x00, BOLD = 0x01, DIM = 0x02, ITALIC = 0x04,
  UNDERLINE = 0x08, _RESERVED_1 = 0x10, INVERSE = 0x20, _RESERVED_2 = 0x40,
  STRIKEOUT = 0x80 }

  Bits for the per-character attributes, which control text features such as italic, bold, underlined text, etc.

- enum CharFlags {
  FG_XTERM = 0x01, BG_XTERM = 0x02, EOL = 0x04,
  RESV_A = 0x08, RESV_B = 0x10, RESV_C = 0x20, RESV_D = 0x40, RESV_E = 0x80,
  COLORMASK = (FG_XTERM | BG_XTERM) }

  Per-character 8 bit flags (uchar) used to manage special states for characters.

- enum OutFlags { OFF = 0x00, CR_TO_LF = 0x01, LF_TO_CR = 0x02, LF_TO_CRLF = 0x04 }

  Output translation flags for special control character translations.

- enum RedrawStyle { NO_REDRAW = 0, RATE_LIMITED, PER_WRITE }

  Determines when Fl_Terminal calls redraw() if new text is added.

Public Member Functions

- void ansi (bool val)
  Enable/disable the ANSI mode flag.

- bool ansi (void) const
  Return the state of the ANSI flag.

- void append (const char ∗s, int len=-1)
  Appends string s to the terminal at the current cursor position using the current text color/attributes.

- void append_ascii (const char ∗s)
  Append NULL terminated ASCII string to terminal, slightly more efficient than append_utf8().

- void append_utf8 (const char ∗buf, int len=-1)
  Append NULL terminated UTF-8 string to terminal.

- void box (Fl_Boxtype val)
  Sets the box type, updates terminal margins et al.

- Fl_Boxtype box (void) const
  Returns the current box type.

- void clear (Fl_Color val)
  Clears the screen to a specific color val and homes the cursor.

- void clear (void)
  Clears the screen to the current textbgcolor(), and homes the cursor.

- void clear_history (void)
  Clears the scroll history buffer and adjusts scrollbar, forcing it to redraw().

- void clear_screen (bool scroll_to_hist=true)
  Clear the terminal screen only; does not affect the cursor position.

- void clear_screen_home (bool scroll_to_hist=true)
  Clear the terminal screen and home the cursor.

- void color (Fl_Color val)
  Sets the background color for the terminal's Fl_Group::box().

- Fl_Color color (void) const
  Return base widget Fl_Group's box() color()

- int cursor_col (void) const
  Return the cursor's current column position on the screen.

- void cursor_home (void)
  Move cursor to the home position (top/left).

- int cursor_row (void) const
  Return the cursor's current row position on the screen.
• void **cursorbgcolor** (Fl_Color val)
  
  Set the cursor's background color used for the cursor itself.

• **Fl_Color** **cursorbgcolor** (void) const

  Get the cursor's background color used for the cursor itself.

• void **cursorfgcolor** (Fl_Color val)

  Set the cursor's foreground color used for text under the cursor.

• **Fl_Color** **cursorfgcolor** (void) const

  Get the cursor's foreground color used for text under the cursor.

• void **display_columns** (int val)

  Set terminal's display width in columns of text characters.

• int **display_columns** (void) const

  Return terminal's display width in columns of text characters.

• void **display_rows** (int val)

  Set terminal's display height in lines of text (rows).

• int **display_rows** (void) const

  Return terminal's display height in lines of text (rows).

• void **draw** (void) FL_OVERRIDE

  Draws the entire Fl_Terminal.

• Fl_Terminal (int X, int Y, int W, int H, const char ∗L, int rows, int cols, int hist)

  Same as the default FLTK constructor, but lets the user force the rows, columns and history to specific sizes on creation.

• Fl_Terminal (int X, int Y, int W, int H, const char ∗L=0)

  The constructor for Fl_Terminal.

• int **handle** (int e) FL_OVERRIDE

  Handle FLTK events.

• void **history_lines** (int val)

  Set the number of lines of screen history.

• int **history_lines** (void) const

  Return the number of lines of screen history.

• void **history_rows** (int val)

  Set terminal's scrollback history buffer size in lines of text (rows).

• int **history_rows** (void) const

  Return terminal's scrollback history buffer size in lines of text (rows).

• int **history_use** (void) const

  Returns how many lines are "in use" by the screen history buffer.

• void **margin_bottom** (int val)

  Set the bottom margin; see Margins.

• int **margin_bottom** (void) const

  Return the bottom margin; see Margins.

• void **margin_left** (int val)

  Set the left margin; see Margins.

• int **margin_left** (void) const

  Return the left margin; see Margins.

• void **margin_right** (int val)

  Set the right margin; see Margins.

• int **margin_right** (void) const

  Return the right margin; see Margins.

• void **margin_top** (int val)

  Set the top margin; see Margins.

• int **margin_top** (void) const

  Return the top margin; see Margins.
• **void output_translate (Fl_Terminal::OutFlags val)**
  Sets the combined output translation flags to val.

• **Fl_Terminal::OutFlags output_translate (void) const**
  Return the current combined output translation flags.

• **void print_char (char c)**
  Prints single ASCII char c at current cursor position, and advances the cursor.

• **void print_char (const char *text, int len=-1)**
  Prints single UTF-8 char text of optional byte length len at current cursor position, and advances the cursor if the character is printable.

• **void printf (const char *fmt,...)**
  Appends printf formatted messages to the terminal.

• **void putchar (char c, int drow, int dcol)**
  Print the ASCII character c at the terminal's display position (drow,dcol).

• **void putchar (const char *text, int len, int drow, int dcol)**
  Print UTF-8 character text of length len at display position (drow,dcol).

• **void redraw_rate (float val)**
  Set the maximum rate redraw speed in floating point seconds if redraw_style() is set to RATE_LIMITED.

• **float redraw_rate (void) const**
  Get max rate redraw speed in floating point seconds.

• **void redraw_style (RedrawStyle val)**
  Set how Fl_Terminal manages screen redrawing.

• **RedrawStyle redraw_style (void) const**
  Get the redraw style.

• **void reset_terminal (void)**
  Resets terminal to default colors, clears screen, history and mouse selection, homes cursor, resets tabstops.

• **void resize (int X, int Y, int W, int H) FL_OVERRIDE**
  Handle widget resizing, such as if user resizes parent window.

• **int scrollbar_actual_size (void) const**
  Returns the scrollbar's actual size; actual width for vertical scrollbars, actual height for horizontal scrollbars.

• **void scrollbar_size (int val)**
  Set the width of the scrollbar's trough to val, in pixels.

• **int scrollbar_size (void) const**
  Get the current size of the scrollbar's trough, in pixels.

• **void selectionbgcolor (Fl_Color val)**
  Set mouse selection background color.

• **Fl_Color selectionbgcolor (void) const**
  Get mouse selection background color.

• **void selectionfgcolor (Fl_Color val)**
  Set mouse selection foreground color.

• **Fl_Color selectionfgcolor (void) const**
  Get mouse selection foreground color.

• **void show_unknown (bool val)**
  Set the "show unknown" flag.

• **bool show_unknown (void) const**
  Return the "show unknown" flag.

• **uchar textattrib () const**
  Get text attribute bits (underline, inverse, etc).

• **void textattrib (uchar val)**
  Set text attribute bits (underline, inverse, etc).

• **void textbgcolor (Fl_Color val)**
  Set text background color to fltk color val used by any new text added.
• **FL_Color** `textbgcolor` (void) const

  Return text’s current background color.

• void `textbgcolor_default (FL_Color val)`

  Set the default text background color used by any new text added after a reset (`<ESC>c, <ESC>[0m`, or `reset_terminal()`).

• **FL_Color** `textbgcolor_default` (void) const

  Return text’s default background color.

• void `textbgcolor_xterm (uchar val)`

  Sets the background text color as one of the 8 ‘xterm color’ values.

• void `textcolor (FL_Color val)`

  Set the text color for the terminal.

• **FL_Color** `textcolor` (void) const

  Return `textcolor()`. This is a convenience method that returns `textfgcolor_default()`.

• void `textfgcolor (FL_Color val)`

  Set text foreground drawing color to fltk color `val` used by any new text added.

• **FL_Color** `textfgcolor` (void) const

  Return text’s current foreground color.

• void `textfgcolor_default (FL_Color val)`

  Set the default text foreground color used by `<ESC>c, <ESC>[0m, and `reset_terminal()`.

• **FL_Color** `textfgcolor_default` (void) const

  Return text’s default foreground color.

• void `textfgcolor_xterm (uchar val)`

  Sets the foreground text color as one of the 8 ‘xterm color’ values.

• void `textfont (FL_Font val)`

  Sets the font used for all text displayed in the terminal.

• **FL_Font** `textfont` (void) const

  Return text font used to draw all text in the terminal.

• void `textsize (FL_Fontsize val)`

  Sets the font size used for all text displayed in the terminal.

• **FL_Fontsize** `textsize` (void) const

  Return text font size used to draw all text in the terminal.

• void `vprintf (const char ∗ fmt, va_list ap)`

  Appends printf formatted messages to the terminal.

• ~`Fl_Terminal` (void)

  The destructor for `Fl_Terminal`.

**Public Member Functions inherited from Fl_Group**

• **FL_Widget** ∗& _ddfdesign_kludge ()

  This is for forms compatibility only.

• void `add (FL_Widget &)`

  The widget is removed from its current group (if any) and then added to the end of this group.

• void `add (FL_Widget ∗)`

  See void Fl_Group::add(Fl_Widget &w)

• void `add_resizable (FL_Widget &o)`

  Adds a widget to the group and makes it the resizable widget.

• **FL_Widget** ∗`const ∗ array () const`

  Returns a pointer to the array of children.

• Fl_Group `const ∗ as_group () const FL_OVERRIDE`

  Returns an Fl_Group pointer if this widget is an Fl_Group.
• void **begin** ()
  
  Sets the current group so you can build the widget tree by just constructing the widgets.

• **Fl_Widget** ∗ **child** (int n) const
  
  Returns `array()[n].`

• int **children** () const
  
  Returns how many child widgets the group has.

• void **clear** ()
  
  Deletes all child widgets from memory recursively.

• unsigned int **clip_children** ()
  
  Returns the current clipping mode.

• void **clip_children** (int c)
  
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

• virtual int **delete_child** (int n)
  
  Removes the widget at *index* from the group and deletes it.

• void **end** ()
  
  Exactly the same as current(this->parent()).

• int **find** (const Fl_Widget &o) const
  
  See int Fl_Group::find(const Fl_Widget ∗w) const.

• int **find** (const Fl_Widget ∗w) const
  
  Searches the child array for the widget and returns the index.

• **Fl_Group** (int, int, int, int, const char ∗=0)
  
  Creates a new Fl_Group widget using the given position, size, and label string.

• void **focus** (Fl_Widget ∗W)

• void **forms_end**()
  
  This is for forms compatibility only.

• void **init_sizes** ()
  
  Resets the internal array of widget sizes and positions.

• void **insert** (Fl_Widget &, int i)
  
  The widget is removed from its current group (if any) and then inserted into this group.

• void **insert** (Fl_Widget &o, Fl_Widget ∗before)
  
  This does insert(w, find(before)).

• void **remove** (Fl_Widget &)
  
  Removes a widget from the group but does not delete it.

• void **remove** (Fl_Widget ∗o)
  
  Removes the widget ∗o from the group.

• void **remove** (int index)
  
  Removes the widget at *index* from the group but does not delete it.

• **Fl_Widget** ∗ **resizable** () const
  
  Returns the group's resizable widget.

• void **resizable** (Fl_Widget &o)
  
  Sets the group's resizable widget.

• void **resizable** (Fl_Widget ∗o)
  
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

• virtual ∼**Fl_Group** ()
  
  The destructor also deletes all the children.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ as_gl_window () const
- virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ as_window () const
- void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
  Checks if the widget value changed since the last callback.
- void clear_active ()
Marks the widget as inactive without sending events or changing focus.

- `void clear_changed ()`
  Marks the value of the widget as unchanged.

- `void clear_damage (uchar c=0)`
  Clears or sets the damage flags.

- `void clear_output ()`
  Sets a widget to accept input.

- `void clear_visible ()`
  Hides the widget.

- `void clear_visible_focus ()`
  Disables keyboard focus navigation with this widget.

- `Fl_Color color () const`
  Gets the background color of the widget.

- `void color (Fl_Color bg)`
  Sets the background color of the widget.

- `void color (Fl_Color bg, Fl_Color sel)`
  Sets the background and selection color of the widget.

- `Fl_Color color2 () const`
  For back compatibility only.

- `void color2 (unsigned a)`
  For back compatibility only.

- `int contains (const Fl_Widget ∗w) const`
  Checks if w is a child of this widget.

- `void copy_label (const char ∗new_label)`
  Sets the current label.

- `void copy_tooltip (const char ∗text)`
  Sets the current tooltip text.

- `uchar damage () const`
  Returns non-zero if draw() needs to be called.

- `void damage (uchar c)`
  Sets the damage bits for the widget.

- `void damage (uchar c, int x, int y, int w, int h)`
  Sets the damage bits for an area inside the widget.

- `int damage_resize (int, int, int, int)`
  Internal use only.

- `void deactivate ()`
  Deactivates the widget.

- `Fl_Image ∗deimage ()`
  Gets the image that is used as part of the widget label when in the inactive state.

- `const Fl_Image ∗deimage () const`
  Gets the image that is used as part of the widget label when in the inactive state.

- `void deimage (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the inactive state.

- `void deimage (Fl_Image ∗img)`
  Sets the image to use as part of the widget label when in the inactive state.

- `int deimage_bound () const`
  Returns whether the inactive image is managed by the widget.

- `void do_callback (FL_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with default arguments.

- `void do_callback (Fl_Widget ∗widget, long arg, FL_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.
void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
Calls the widget callback function with arbitrary arguments.

void draw_label (int, int, int, int, Fl_Align) const
Draws the label in an arbitrary bounding box with an arbitrary alignment.

int h () const
Gets the widget height.

virtual void hide ()
Makes a widget invisible.

Fl_Image * image ()
Gets the image that is used as part of the widget label when in the active state.

const Fl_Image * image () const
Gets the image that is used as part of the widget label when in the active state.

void image (Fl_Image &img)
Sets the image to use as part of the widget label when in the active state.

void image (Fl_Image *img)
Sets the image to use as part of the widget label when in the active state.

int image_bound () const
Returns whether the image is managed by the widget.

int inside (const Fl_Widget *wgt) const
Checks if this widget is a child of wgt.

int is_label_copied () const
Returns whether the current label was assigned with copy_label().

const char * label () const
Gets the current label text.

void label (const char *text)
Sets the current label pointer.

void label (Fl_Labeltype a, const char *b)
Shortcut to set the label text and type in one call.

Fl_Color labelcolor () const
Gets the label color.

void labelcolor (Fl_Color c)
Sets the label color.

Fl_Font labelfont () const
Gets the font to use.

void labelfont (Fl_Font f)
Sets the font to use.

Fl_Fontsize labelsize () const
Gets the font size in pixels.

void labelsize (Fl_Fontsize pix)
Sets the font size in pixels.

Fl_Labeltype labeltype () const
Gets the label type.

void labeltype (Fl_Labeltype a)
Sets the label type.

void measure_label (int &ww, int &hh) const
Sets width ww and height hh accordingly with the label size.

bool needs_keyboard () const
Returns whether this widget needs a keyboard.

void needs_keyboard (bool needs)
Sets whether this widget needs a keyboard.

unsigned int output () const
1574 Class Documentation

Returns if a widget is used for output only.

- **Fl_Group** → **parent ()** const
  Returns a pointer to the parent widget.

- **void parent (Fl_Group *p)**
  Internal use only - "for hacks only".

- **void position (int X, int Y)**
  Repositions the window or widget.

- **void redraw ()**
  Schedules the drawing of the widget.

- **void redraw_label ()**
  Schedules the drawing of the label.

- **Fl_Color selection_color ()** const
  Gets the selection color.

- **void selection_color (Fl_Color a)**
  Sets the selection color.

- **void set_active ()**
  Marks the widget as active without sending events or changing focus.

- **void set_changed ()**
  Marks the value of the widget as changed.

- **void set_output ()**
  Sets a widget to output only.

- **void set_visible ()**
  Makes the widget visible.

- **void set_visible_focus ()**
  Enables keyboard focus navigation with this widget.

- **int shortcut_label ()** const
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.

- **void shortcut_label (int value)**
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.

- **virtual void show ()**
  Makes a widget visible.

- **void size (int W, int H)**
  Changes the size of the widget.

- **int take_focus ()**
  Gives the widget the keyboard focus.

- **unsigned int takesevents ()** const
  Returns if the widget is able to take events.

- **int testShortcut ()**
  Returns true if the widget’s label contains the entered ‘&x’ shortcut.

- **const char * tooltip ()** const
  Gets the current tooltip text.

- **void tooltip (const char *text)**
  Sets the current tooltip text.

- **Fl_Window * top_window ()** const
  Returns a pointer to the top-level window for the widget.

- **Fl_Window * top_window_offset (int &xoff, int &yoff) const**
  Finds the x/y offset of the current widget relative to the top-level window.

- **uchar type ()** const
  Gets the widget type.

- **void type (uchar t)**
  Sets the widget type.
• int **use_accents_menu** ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void **user_data** () const
  Gets the user data for this widget.

• void **user_data** (Fl_Callback_User_Data \*v, bool auto_free)
  Sets the user data for this widget.

• void **user_data** (void \*v)
  Sets the user data for this widget.

• unsigned int **visible** () const
  Returns whether a widget is visible.

• unsigned int **visible_focus** () const
  Checks whether this widget has a visible focus.

• void **visible_focus** (int v)
  Modifies keyboard focus navigation.

• int **visible_r** () const
  Returns whether a widget and all its parents are visible.

• int **w** () const
  Gets the widget width.

• Fl_When **when** () const
  Returns the conditions under which the callback is called.

• void **when** (uchar i)
  Sets the flags used to decide when a callback is called.

• Fl_Window **window** () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int **x** () const
  Gets the widget position in its window.

• int **y** () const
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  Destroys the widget.

Public Attributes

• Fl_Scrollbar \* **scrollbar**
  Vertical scrollbar.

Protected Member Functions

• void **clear_eod** (void)
  Clear from cursor to End Of Display (EOD), like *<ESC>\[J<ESC>\[0J*.

• void **clear_eol** (void)
  Clear from cursor to End Of Line (EOL), like *<ESC>\[K*.

• void **clear_line** (int row)
  Clear entire line for specified row.

• void **clear_line** (void)
  Clear entire line cursor is currently on.

• void **clear_mouse_selection** (void)
  Clear any current mouse selection.

• void **clear_sod** (void)
  Clear from cursor to Start Of Display (EOD), like *<ESC>\[1J*.

• void **clear_sol** (void)
  Clear from cursor to Start Of Line (SOL), like *<ESC>\[1K*.
- **void** **current_style** (const CharStyle &sty)
  
  Set current style for rendering text.

- **const CharStyle &** **current_style** (void) const
  
  Return current style for rendering text.

- **void** **cursor_col** (int col)
  
  Move cursor to the specified column col.

- **void** **cursor_cr** (void)
  
  Move cursor as if a CR (\r) was received.

- **void** **cursor_crlf** (int count=1)
  
  Move cursor as if a CR/LF pair (\r\n) was received.

- **void** **cursor_down** (int count=1, bool do_scroll=false)
  
  Moves cursor down count lines.

- **void** **cursor_eol** (void)
  
  Move cursor to the last column (at the far right) on the current line.

- **void** **cursor_left** (int count=1)
  
  Moves cursor left count columns, and cursor stops (does not wrap) if it hits screen edge.

- **void** **cursor_right** (int count=1, bool do_scroll=false)
  
  Moves cursor right count columns.

- **void** **cursor_row** (int row)
  
  Move cursor to the specified row row.

- **void** **cursor_sol** (void)
  
  Move cursor to the first column (at the far left) on the current line.

- **void** **cursor_tab_left** (int count=1)
  
  Tab left, do not wrap beyond left edge.

- **void** **cursor_tab_right** (int count=1)
  
  Tab right, do not wrap beyond right edge.

- **void** **cursor_up** (int count=1, bool do_scroll=false)
  
  Moves cursor up count lines.

- **void** **delete_chars** (int drow, int dcol, int rep)
  
  Delete char(s) at (drow,dcol) for 'rep' times.

- **void** **delete_chars** (int rep)
  
  Delete char(s) at cursor position for 'rep' times.

- **void** **delete_rows** (int count)
  
  Delete (count) rows at cursor position.

- **int** **disp_cols** (void) const
  
  Return the number of columns in the display area.

- **int** **disp_erow** (void) const
  
  Return the ending row# in the display area.

- **int** **disp_rows** (void) const
  
  Return the number of rows in the display area.

- **int** **disp_srow** (void) const
  
  Return the starting row# in the display area.

- **int** **draw_buff** (int Y) const
  
  Draws the buffer position we are scrolled to onto the FLTK screen starting at pixel position Y.

- **void** **draw_row** (int grow, int int Y) const
  
  Draw the specified global row, which is the row in ring_chars[].

- **void** **draw_row_bg** (int grow, int X, int Y) const
  
  Draw the background for the specified ring_chars[] global row grow starting at FLTK coords X and Y.

- **bool** **get_selection** (int &srow, int &scol, int &erow, int &ecol) const
  
  Return mouse selection's start/end position in the ring buffer, if any.

- **int** **h_to_row** (int H) const
Given a height in pixels, return number of rows that "fits" into that area.

- int handle_unknown_char (void)
  Handle an unknown char by either emitting an error symbol to the tty, or do nothing, depending on the user configurable value of show_unknown().

- int hist_cols (void) const
  Return the number of columns in the scrollback history.

- int hist_erow (void) const
  Return the ending row# of the scrollback history.

- int hist_rows (void) const
  Return the number of rows in the scrollback history.

- int hist_srow (void) const
  Return the starting row# of the scrollback history.

- int hist_use (void) const
  Return number of rows in use by the scrollback history.

- int hist_use_srow (void) const
  Return the starting row of the "in use" scrollback history.

- void insert_char (char c, int rep)
  Insert char 'c' at the current cursor position for 'rep' times.

- void insert_char_eol (char c, int drow, int dcol, int rep)
  Insert char 'c' for 'rep' times at display row 'drow' and column 'dcol'.

- void insert_rows (int count)
  Insert (count) rows at current cursor position.

- bool is_inside_selection (int row, int col) const
  Is global row/column inside the current mouse selection?

- bool is_selection (void) const
  Returns true if there's a mouse selection.

- int offset (void) const
  Returns the current offset into the ring buffer.

- void restore_cursor (void)
  Restore previously saved cursor position, if any. Used by ESC [ u.

- int ring_cols (void) const
  Return the number of columns in the ring buffer.

- int ring_erow (void) const
  Return the ending row# in the ring buffer (Always ring_rows()-1)

- int ring_rows (void) const
  Return the number of rows in the ring buffer.

- int ring_srow (void) const
  Return the starting row# in the ring buffer. (Always 0)

- void save_cursor (void)
  Save current cursor position. Used by ESC [ s.

- void scroll (int rows)
  Scroll the display up(+) or down(-) the specified rows.

- int scrollbar_width (void) const
  Return scrollbar width if visible, or 0 if not visible.

- bool selection_extend (int X, int Y)
  Extend selection to FLTK coords X,Y.

- const char * selection_text (void) const
  Return text selection (for copy()/paste() operations)

- int selection_text_len (void) const
  Return byte length of all UTF-8 chars in selection, or 0 if no selection.

- U8Char * u8c_cursor (void)
Return the Utf8Char for character under cursor.

- Utf8Char * u8c_disp_row (int drow)
  Return pointer to the first u8c character in row drow of the display.

- const Utf8Char * u8c_disp_row (int drow) const
  See docs for non-const version of u8c_disp_row(int)

- Utf8Char * u8c_hist_row (int hrow)
  Return u8c for beginning of a row inside the scrollback history.

- const Utf8Char * u8c_hist_row (int hrow) const
  See docs for non-const version of u8c_hist_row(int)

- const Utf8Char * u8c_hist_use_row (int hrow) const
  See docs for non-const version of u8c_hist_use_row(int)

- Utf8Char * u8c_ring_row (int grow)
  Return UTF-8 char for row grow in the ring buffer.

- const Utf8Char * u8c_ring_row (int grow) const
  See docs for non-const version of u8c_ring_row(int)

- int w_to_col (int W) const
  Given a width in pixels, return number of columns that "fits" into that area.

- const Utf8Char * walk_selection (const Utf8Char *u8c, int &row, int &col) const
  Walk the mouse selection one character at a time from beginning to end, returning a Utf8Char* to the next character in the selection, or NULL if the end was reached, or if there's no selection.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.

- void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.

- void draw_children ()
  Draws all children of the group.

- void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.

- virtual int on_insert (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.

- virtual int on_move (int, int)
  Allow derived groups to act when a widget is moved within the group.

- virtual void on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.

- int * sizes ()
  Returns the internal array of widget sizes and positions.

- void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.

- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void draw_box () const
  Draws the widget box according its box style.
- `void draw_box` (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

- `void draw_box` (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- `void draw_focus` () const
  Draws a focus rectangle around the widget.

- `void draw_focus` (Fl_Boxtype t, int x, int y, int w, int h) const
  Draws a focus rectangle around the widget.

- `void draw_focus` (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

- `void draw_label` () const
  Draws the widget's label at the defined label position.

- `void draw_label` (int, int, int, int) const
  Draws the label in an arbitrary bounding box.

- `Fl_Widget` (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.

- `unsigned int flags` () const
  Gets the widget flags mask.

- `void h` (int v)
  Internal use only.

- `void set_flag` (unsigned int c)
  Sets a flag in the flags mask.

- `void w` (int v)
  Internal use only.

- `void x` (int v)
  Internal use only.

- `void y` (int v)
  Internal use only.

### Additional Inherited Members

### Static Public Member Functions inherited from `Fl_Group`

- static `Fl_Group ∗current` ()
  Returns the currently active group.

- static `void current` (Fl_Group ∗g)
  Sets the current group.

### Static Public Member Functions inherited from `Fl_Widget`

- static `void default_callback` (Fl_Widget ∗widget, void ∗data)
  The default callback for all widgets that don't set a callback.

- static `unsigned int label_shortcut` (const char ∗t)
  Returns the Unicode value of the '&x' shortcut in a given text.

- static `int test_shortcut` (const char ∗, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.
Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,  
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7, 
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, Copied_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,  
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,  
  GROUP_RELATIVE = 1<<16, Copied_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,  
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,  
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31}

flags possible values enumeration.

12.141.1 Detailed Description

Terminal widget supporting Unicode/utf-8, ANSI/xterm escape codes with full RGB color control.

12.141.2 Fl_Terminal

![Figure 12.54 Fl_Terminal widget showing a linux manual page](image)

Fl_Terminal is an output-only text widget supporting ASCII and UTF-8/Unicode. It supports most terminal text features, such as most VT100/xterm style escape sequences (see The Escape Codes Fl_Terminal Supports), text colors/attributes, scrollback history, mouse selection, etc.

It is recommended that accessing features such as setting text colors and cursor positioning is best done with ANSI/XTERM escape sequences. But if one sets ansi(false) then this is not possible, so the public API can be used for common operations, e.g.

<table>
<thead>
<tr>
<th>Public API</th>
<th>ESC code equivalent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear_screen_home()</td>
<td>ESC [ H ESC [ 2 J</td>
<td>Clear screen, home cursor</td>
</tr>
<tr>
<td>cursor_home()</td>
<td>ESC [ H</td>
<td>Home the cursor</td>
</tr>
<tr>
<td>clear_history()</td>
<td>ESC [ 3 J</td>
<td>Clear scrollback history</td>
</tr>
<tr>
<td>reset_terminal()</td>
<td>ESC [ c</td>
<td>Reset terminal</td>
</tr>
</tbody>
</table>

To access more advanced API calls, one can derive a class from Fl_Terminal to access protected methods manipulate the terminal more directly, e.g.
### Protected API

<table>
<thead>
<tr>
<th>Function</th>
<th>ESC code equiv.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_style()</td>
<td>ESC [ # m</td>
<td>Set text attributes</td>
</tr>
<tr>
<td>clear_eod()</td>
<td>ESC [ 0 J</td>
<td>Clear from cursor to end of display</td>
</tr>
<tr>
<td>clear_sod()</td>
<td>ESC [ 1 J</td>
<td>Clear from cursor to start of display</td>
</tr>
<tr>
<td>clear_eol()</td>
<td>ESC [ 0 K</td>
<td>Clear from cursor to end of line</td>
</tr>
<tr>
<td>clear_sol()</td>
<td>ESC [ 1 K</td>
<td>Clear from cursor to start of line</td>
</tr>
<tr>
<td>clear_line()</td>
<td>ESC [ 2 K</td>
<td>Clear line cursor is on</td>
</tr>
<tr>
<td>scroll(int)</td>
<td>ESC [ 1 S</td>
<td>Scroll up one line</td>
</tr>
<tr>
<td>scroll(int)</td>
<td>ESC [ 1 T</td>
<td>Scroll down one line</td>
</tr>
<tr>
<td>cursor_left()</td>
<td>ESC [ 1 D</td>
<td>Move cursor left (no wrap)</td>
</tr>
<tr>
<td>cursor_right()</td>
<td>ESC [ 1 C</td>
<td>Move cursor right (no wrap)</td>
</tr>
<tr>
<td>cursor_up()</td>
<td>ESC [ 1 B</td>
<td>Move cursor up (no scroll or wrap)</td>
</tr>
<tr>
<td>cursor_down()</td>
<td>ESC [ 1 A</td>
<td>Move cursor down (no scroll or wrap)</td>
</tr>
<tr>
<td>cursor_row()</td>
<td>ESC [ # ; # H</td>
<td>Move cursor to row# / column#</td>
</tr>
<tr>
<td>cursor_col()</td>
<td>ESC [ # ; # H</td>
<td>Move cursor to row# / column#</td>
</tr>
<tr>
<td>insert_char()</td>
<td>ESC [ # @</td>
<td>Insert a char at cursor position</td>
</tr>
<tr>
<td>delete_chars()</td>
<td>ESC [ # P</td>
<td>Delete chars at cursor position</td>
</tr>
<tr>
<td>insert_rows()</td>
<td>ESC [ # L</td>
<td>Insert rows at cursor position</td>
</tr>
<tr>
<td>delete_rows()</td>
<td>ESC [ # M</td>
<td>Delete rows at cursor position</td>
</tr>
<tr>
<td>etc...</td>
<td>etc...</td>
<td>etc...</td>
</tr>
</tbody>
</table>

Many commonly used API functions are public, such as `textfgcolor()` for setting text colors. Others, such as `cursor_up()` are protected to prevent common misuse, and are available only to subclasses.

For applications that need input support, the widget can be subclassed to provide keyboard input, and advanced features like pseudo ttys, termio, serial port I/O, etc., as such features are beyond the scope of FLTK.

#### 12.141.2.1 Examples

```c++
//
// Fl_Terminal: Simple Use
//
Fl_Terminal *tty = new Fl_Terminal(...);

tty->append("Hello world.\n"); // simple strings
tty->append("\033[31mThis text is red.\033[0m\n"); // colored text
tty->printf("The value of x is %.02f\n", x); // printf() formatting

tty->home(); // home the cursor
tty->clear_screen(); // clear the screen
tty->textfgcolor(0xff000000); // change the text color to RED
tty->textbgcolor(0x0000ff00); // change the background color to BLUE

// R G B
```

There are also public methods for doing what most "\033[" escape codes do, so that if `ansi(bool)` is set to "false", one can still change text colors or clear the screen via application control, e.g.

```c++
tty->home(); // home the cursor
tty->clear_screen(); // clear the screen
```

When creating the widget, the width/height determine the default column and row count for the terminal's display based on the current font size. The column width determines where text will wrap.

You can specify wider column sizes than the screen using `display_columns(colwidth)`. When this value is larger than the widget's width, text will wrap off-screen, and can be revealed by resizing the widget wider.
12.141.2.2 Writing To Terminal From Applications

An application needing terminal output as part of its user interface can instance `Fl_Terminal`, and write text strings with:

- `append()` to append strings
- `printf()` to append formatted strings

Single character output can be done with:

- `print_char()` to print a single ASCII/UTF-8 char at the cursor
- `putchar()` to put single ASCII/UTF-8 char at an x,y position

12.141.2.3 Text Attributes

The terminal's text supports these attributes:

- Italic - italicized text: `\033[3m`
- Bold - brighter/thicker text: `\033[1m`
- Dim - lower brightness text: `\033[2m`
- Underline - text that is underlined: `\033[4m`
- Strikeout - text that has a line through the text: `\033[9m`
- Inverse - text whose background and foreground colors are swapped: `\033[7m`
- Normal - normal text: `\033[0m`

12.141.2.4 Text and Background Colors

There's at least two ways to specify colors for text and background colors:

- 3 bit / 8 Color Values
- Full 24 bit R/G/B colors
12.141 Fl_Terminal Class Reference

Example of 3 bit colors:

![Figure 12.56 Fl_Terminal 3 bit colors](image)

Example application source code using 3 bit colors:
```
// Text colors
tty->append("\033[31m Red text. \033[0m\n"); // Print red text..
```
```
// Background colors
tty->append("\033[41m Red Background. \033[0m\n"); // background will be red
```

Example of 24 bit colors:
```
// 24 bit Text Color
tty->append("\033[38;2;0;0;255m Text is BLUE. \033[0m\n"); // RGB: R=0, G=0, B=255
```
```
// 24 bit Background Color
tty->append("\033[48;2;0;0;255m Background is BLUE. \033[0m\n"); // RGB: R=0, G=0, B=255
```

For more on the ANSI escape codes, see The Escape Codes Fl_Terminal Supports.
12.141.2.5 Features

Most standard terminal behaviors are supported, e.g.

- ASCII + UTF-8/Unicode
- scrollback history management
- mouse selection + copy/paste (\^C, \^A)
- autoscroll during selection

Most popular ANSI/DEC VT100/Xterm escape sequences are supported (see The Escape Codes Fl_Terminal Supports), including:

- per-character colors for text and background
- per-character text attributes: bold/dim, underline, strikeout
- scrolling up/down
- character insert/delete for characters/rows/screen
- clearing characters/rows/screen

Does not (yet) support:

- programmable regions (scroll regions and attribute blocks)
- dynamic line wrap (where resizing display dynamically re-wraps long lines)

Will likely never implement as part of this widget:

- pty/termio management (such features should be subclassed)
- Different per-character font family + sizes (font family/size is global only)
- variable width fonts

Regarding the font family+size; the way the terminal is currently designed, the font family and size must not vary within text; rows have to be consistent height. Varying widths are tricky too, esp. when it comes to moving the cursor up/down within a column; varying widths are supported (due to Unicode characters sometimes being "wide", but not heights.

12.141.2.6 Margins

The margins define the amount of space (in pixels) around the outside of the text display area, the space between the widget's inner edge (inside the box()) and the text display area's outer edge. The margins can be inspected and changed with the margin_left(), margin_right(), margin_top() and margin_bottom() methods.
TERMINAL WIDGET (Fl_Terminal)

TEXT DISPLAY AREA

s c
display_rows r
c ol l
display_columns b a
r

margin_left ↑ margin_top
margin_bottom ↓ margin_right

Fl_Terminal Margins

::##:::##:::
::###:::###::
::####:####:: color() color()
::##:###:##::
::##::#::##::
::##:.:::##::
::##:::::#::
textbgcolor() textbgcolor()

12.141.2.7 Caveats

• This widget is not a full terminal emulator; it does not do stdio redirection, pseudo ttys/termios/character cooking, keyboard input processing, full curses(3) support. However, such features CAN be implemented with subclassing.

• The printf() and vprintf() functions are limited to strings no longer than 1024 characters (including NULL). For printing longer strings, use append() which has no string limits.

• For background colors textbgcolor() and textbgcolor_default(), the special color value 0xffffffff lets the widget's box() color() show through behind the text.

12.141.3 Member Enumeration Documentation

12.141.3.1 Attrib

enum Fl_Terminal::Attrib

Bits for the per-character attributes, which control text features such as italic, bold, underlined text, etc.

NOTE: enum names with a leading underbar must not be used, and are reserved for future use.

 Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL</td>
<td>all attributes off</td>
</tr>
<tr>
<td>BOLD</td>
<td>bold text: uses bold font, color brighter than normal</td>
</tr>
<tr>
<td>DIM</td>
<td>dim text; color slightly darker than normal</td>
</tr>
<tr>
<td>ITALIC</td>
<td>italic font text</td>
</tr>
</tbody>
</table>
### 12.141.3.2 CharFlags

`enum Fl_Terminal::CharFlags`

Per-character 8 bit flags (uchar) used to manage special states for characters.

#### Enumerator

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FG_XTERM</td>
<td>this char's fg color is an XTERM color; can be affected by Dim+Bold</td>
</tr>
<tr>
<td>BG_XTERM</td>
<td>this char's bg color is an XTERM color; can be affected by Dim+Bold</td>
</tr>
<tr>
<td>EOL</td>
<td>TODO: char at EOL, used for line re-wrap during screen resizing.</td>
</tr>
</tbody>
</table>

### 12.141.3.3 OutFlags

`enum Fl_Terminal::OutFlags`

Output translation flags for special control character translations.

#### Enumerator

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>no output translation</td>
</tr>
<tr>
<td>CR_TO_LF</td>
<td>carriage return generates a vertical line-feed (\r -&gt; \n)</td>
</tr>
<tr>
<td>LF_TO_CR</td>
<td>line-feed generates a carriage return (\n -&gt; \r)</td>
</tr>
<tr>
<td>LF_TO_CRLF</td>
<td>line-feed generates a carriage return line-feed (\n -&gt; \r\n)</td>
</tr>
</tbody>
</table>

### 12.141.3.4 RedrawStyle

`enum Fl_Terminal::RedrawStyle`

Determines when `Fl_Terminal` calls `redraw()` if new text is added.

RATE_LIMITED is the recommended setting, using `redraw_rate(float)` to determine the maximum rate of redraws.

See also

`redraw_style()`, `redraw_rate()`

#### Enumerator

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_REDRAW</td>
<td>app must call <code>redraw()</code> as needed to update text to screen</td>
</tr>
<tr>
<td>RATE_LIMITED</td>
<td>timer controlled redraws. (DEFAULT)</td>
</tr>
<tr>
<td>PER_WRITE</td>
<td>redraw triggered after every <code>append()</code> / <code>printf()</code> / etc. operation</td>
</tr>
</tbody>
</table>
12.141.4 Constructor & Destructor Documentation

12.141.4.1 Fl_Terminal() [1/2]

Fl_Terminal::Fl_Terminal (  
    int X,
    int Y,
    int W,
    int H,
    const char * L = 0  )

The constructor for Fl_Terminal.
This creates an empty terminal with defaults:

- white on black text; see textfgcolor(Fl_Color), textbgcolor(Fl_Color)
- rows/cols based on the W and H values, see display_rows(), display_columns()
- scrollback history of 100 lines, see history_rows()
- redraw_style() set to RATE_LIMITED, redraw_rate() set to 0.10 seconds

Note: While Fl_Terminal derives from Fl_Group, it's not intended for user code to use it as a parent for other widgets, so end() is called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X,Y,W,H</th>
<th>position and size.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>label string (optional), may be NULL.</td>
</tr>
</tbody>
</table>

12.141.4.2 Fl_Terminal() [2/2]

Fl_Terminal::Fl_Terminal (  
    int X,
    int Y,
    int W,
    int H,
    const char * L,
    int rows,
    int cols,
    int hist  )

Same as the default FLTK constructor, but lets the user force the rows, columns and history to specific sizes on creation.
Since the row/cols/hist are specified directly, this prevents the widget from auto-calculating the initial text buffer size based on the widget's pixel width/height, bypassing calls to the font system before the widget is displayed.

Note

fluid uses this constructor internally to avoid font calculations that opens the display, useful for when running in a headless context. (issue 837)

12.141.4.3 ~Fl_Terminal()

Fl_Terminal::~Fl_Terminal (  
    void  )

The destructor for Fl_Terminal.
Destroyes the terminal display, scroll history, and associated widgets.
12.141.5 Member Function Documentation

12.141.5.1 ansi() [1/2]

```cpp
void Fl_Terminal::ansi (bool val)
```
Enable/disable the ANSI mode flag.
If true, ANSI and VT100/xterm codes will be processed. If false, these codes won't be processed and will either be ignored or print the error character "¿", depending on the value of show_unknown().

See also
- show_unknown(), The Escape Codes Fl_Terminal Supports

12.141.5.2 ansi() [2/2]

```cpp
bool Fl_Terminal::ansi () const
```
Return the state of the ANSI flag.

See also
- ansi(bool)

12.141.5.3 append()

```cpp
void Fl_Terminal::append (const char ∗s, int len = -1)
```
Appends string `s` to the terminal at the current cursor position using the current text color/attributes.
If `s` is NULL, the UTF-8 character cache is cleared, which is recommended before starting a block reading loop, and again after the block loop has completed.
If `len` is not specified, it's assumed `s` is a NULL terminated string. If `len` is specified, it can be used for writing strings that aren't NULL terminated, such as block reads on a pipe, network, or other block oriented data source.
Redraws of the terminal widget are by default handled automatically, but can be changed with redraw_rate() and redraw_style().

**Block I/O**
When reading block oriented sources (such as pipes), append() will handle partial UTF-8 chars straddling the block boundaries. It does this using an internal byte cache, which should be cleared before and after block I/O loops by calling append(NULL) as shown in the example below, to prevent the possibilities of partial UTF-8 characters left behind by an interrupted or incomplete block loop.

```cpp
// Example block reading a command pipe in Unix

// Run command and read as a pipe
FILE ∗fp = popen("ls -la", "r");
if (!fp) { ..error_handling.. }

// Enable non-blocking I/O
int fd = fileno(fp);
fcntl(fd, F_SETFL, O_NONBLOCK);

// Clear UTF-8 character cache before starting block loop
G_tty->append(NULL); // prevents leftover partial UTF-8 bytes

// Block read loop
while (1) {
    Fl::wait(0.05); // give fltk .05 secs of cpu to manage UI
    ssize_t bytes = read(fd, s, sizeof(s)); // read block from pipe
    if (bytes == -1 && errno == EAGAIN) continue; // no data yet? continue
    if (bytes > 0) G_tty->append(s); // append output to terminal
    else break; // end of pipe?
}

// Flush cache again after block loop completes
G_tty->append(NULL);

// Close pipe, done
pclose(fp);
```
Note

- String can contain ASCII or UTF-8 chars
- len is optional; if unspecified, expects s to be a NULL terminated string
- Handles partial UTF-8 chars split between calls (e.g. block oriented writes)
- If s is NULL, this clears the "partial UTF-8" character cache
- Redraws are managed automatically by default; see redraw_style()

12.141.5.4  append_ascii()

void Fl_Terminal::append_ascii (  
    const char * s )
Append NULL terminated ASCII string to terminal, slightly more efficient than append_utf8().
  
- If s is NULL, behavior is to do nothing
- Redraws are triggered automatically, depending on redraw_style()

12.141.5.5  append_utf8()

void Fl_Terminal::append_utf8 (  
    const char * buf,  
    int len = -1 )
Append NULL terminated UTF-8 string to terminal.
  
- If buf is NULL, UTF-8 cache buffer is cleared
- If optional len isn't specified or is -1, strlen(text) is used.
- If len is 0 or < -1, no changes are made
- Handles UTF-8 chars split across calls (e.g. block writes from pipes, etc)
- Redraws are triggered automatically, depending on redraw_style()

12.141.5.6  box()

void Fl_Terminal::box (  
    Fl_Boxtype val ) [inline]
Sets the box type, updates terminal margins et al.  
Default is FL_DOWN_FRAME.
FL_XXX_FRAME types are handled in a special way by this widget, and guarantee the background is a flat field.
FL_XXX_BOX may draw gradients as inherited by Fl::scheme().

12.141.5.7  clear() [1/2]

void Fl_Terminal::clear (  
    Fl_Color val )
Clears the screen to a specific color val and homes the cursor.
See also  
  clear_screen(), clear_screen_home(), cursor_home()

12.141.5.8  clear() [2/2]

void Fl_Terminal::clear (  
    void )
Clears the screen to the current textbgcolor(), and homes the cursor.
See also  
  clear_screen(), clear_screen_home(), cursor_home()
12.141.5.9  clear_screen()

void Fl_Terminal::clear_screen (  
    bool scroll_to_hist = true )

Clear the terminal screen only; does not affect the cursor position.
Also clears the current mouse selection.
If 'scroll_to_hist' is true, the screen is cleared by scrolling the contents into the scrollback history, where
it can be retrieved with the scrollbar. This is the default behavior. If false, the screen is cleared and the scrollback
history is unchanged.
Similar to the escape sequence "<ESC>[2J".

See also
  clear_screen_home()

12.141.5.10  clear_screen_home()

void Fl_Terminal::clear_screen_home (  
    bool scroll_to_hist = true )

Clear the terminal screen and home the cursor.
Also clears the current mouse selection.
If 'scroll_to_hist' is true, the screen is cleared by scrolling the contents into the scrollback history, where
it can be retrieved with the scrollbar. This is the default behavior. If false, the screen is cleared and the scrollback
history is unchanged.
Similar to the escape sequence "<ESC>[2J<ESC>[H".

See also
  clear_screen()

12.141.5.11  color()

void Fl_Terminal::color (  
    Fl_Color val )

Sets the background color for the terminal's Fl_Group::box().
If the textbgcolor() and textbgcolor_default() are set to the special "see through" color 0xffffffff when any text was
added, changing color() affects the color that shows through behind that existing text.
Otherwise, whatever specific background color was set for existing text will persist after changing color().
To see the effects of a change to color(), follow up with a call to redraw().
The default value is 0x0.

12.141.5.12  cursor_col()

void Fl_Terminal::cursor_col (  
    int col ) [protected]

Move cursor to the specified column col.
This value is clamped to the range (0..display_columns()-1).

12.141.5.13  cursor_cr()

void Fl_Terminal::cursor_cr (  
    void ) [protected]

Move cursor as if a CR (\r) was received.
Same as cursor_sol()

12.141.5.14  cursor_down()

void Fl_Terminal::cursor_down (  
    int count = 1,  
    bool do_scroll = false ) [protected]
Moves cursor down \texttt{count} lines. If cursor hits screen bottom, it either stops (does not wrap) if \texttt{do_scroll} is false, or wraps and scrolls up if \texttt{do_scroll} is true.

**Parameters**

<table>
<thead>
<tr>
<th>count</th>
<th>Number of lines to move cursor down</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{do_scroll}</td>
<td>Enable scrolling if set to true</td>
</tr>
</tbody>
</table>

**12.141.5.15 cursor_right()**

```cpp
void Fl_Terminal::cursor_right(
    int count = 1,
    bool do_scroll = false \} [protected]
```

Moves cursor right \texttt{count} columns. If cursor hits right edge of screen, it either stops (does not wrap) if \texttt{do_scroll} is false, or wraps and scrolls up one line if \texttt{do_scroll} is true.

**12.141.5.16 cursor_row()**

```cpp
void Fl_Terminal::cursor_row(
    int row \) [protected]
```

Move cursor to the specified \texttt{row}. This value is clamped to the range (0..\texttt{display_rows()}-1).

**12.141.5.17 cursor_up()**

```cpp
void Fl_Terminal::cursor_up(
    int count = 1,
    bool do_scroll = false \} [protected]
```

Moves cursor up \texttt{count} lines. If cursor hits screen top, it either stops (does not wrap) if \texttt{do_scroll} is false, or scrolls down if \texttt{do_scroll} is true.

**12.141.5.18 delete_rows()**

```cpp
void Fl_Terminal::delete_rows(
    int count \} [protected]
```

Delete \texttt{count} rows at cursor position. Causes rows to scroll up, and empty lines created at bottom of screen. Lines deleted by scroll up are NOT moved into the scroll history.

**12.141.5.19 display_columns() [1/2]**

```cpp
void Fl_Terminal::display_columns(
    int dcols \)
```

Set terminal’s display width in columns of text characters. This value is normally managed automatically by \texttt{resize()} based on the current font size, and should not be changed. You CAN make the \texttt{display_columns()} larger than the width of the widget; text in the terminal will simply run off the screen edge and be clipped; the only way to reveal that text is if the user enlarges the widget, or the font size made smaller.

To change the display width, it is best to use \texttt{resize()} instead.

**12.141.5.20 display_columns() [2/2]**

```cpp
int Fl_Terminal::display_columns(
    void \} const
```

Return terminal’s display width in columns of text characters.
This value is normally managed automatically by `resize()` based on the current font size.

### 12.141.5.21 display_rows() [1/2]

```cpp
void Fl_Terminal::display_rows ( int drows )
```

Set terminal's display height in lines of text (rows).
This value is normally managed automatically by `resize()` based on the current font size, and should not be changed. To change the display height, use `resize()` instead.

### 12.141.5.22 display_rows() [2/2]

```cpp
int Fl_Terminal::display_rows () const
```

Return terminal's display height in lines of text (rows).
This value is normally managed automatically by `resize()` based on the current font size.

### 12.141.5.23 draw()

```cpp
void Fl_Terminal::draw ( void ) [virtual]
```

Draws the entire Fl_Terminal.
Let the group draw itself first (scrollbar should be only member), followed by the terminal's screen contents.
Reimplemented from Fl_Group.

### 12.141.5.24 draw_buff()

```cpp
void Fl_Terminal::draw_buff ( int Y ) const [protected]
```

Draws the buffer position we are scrolled to onto the FLTK screen starting at pixel position Y.
This can be anywhere in the ring buffer, not just the 'active display'; depends on what position the scrollbar is set to.
Handles attributes, colors, text selections, cursor.

### 12.141.5.25 draw_row()

```cpp
void Fl_Terminal::draw_row ( int grow, int Y ) const [protected]
```

Draw the specified global row, which is the row in ring_chars[].
The global row includes history + display buffers.

### 12.141.5.26 draw_row_bg()

```cpp
void Fl_Terminal::draw_row_bg ( int grow, int X, int Y ) const [protected]
```

Draw the background for the specified ring_chars[] global row `grow` starting at FLTK coords X and Y.
Note we may be called to draw display, or even history if we're scrolled back. If there's any change in bg color, we draw the filled rects here.
If the bg color for a character is the special "see through" color 0xffffffff, no pixels are drawn.

### 12.141.5.27 get_selection()

```cpp
bool Fl_Terminal::get_selection ( int & srow, int & scol, int & erow, int & ecol ) const [protected]
```
Return mouse selection’s start/end position in the ring buffer, if any.
Ensures (start < end) to allow walking ’forward’ thru selection, left-to-right, top-to-bottom. The row/col values are indexes into the entire ring buffer.

Example: walk the characters of the mouse selection:

```c
// Get selection
int srow,scol,erow,ecol;
if (get_selection(srow,scol,erow,ecol)) { // mouse selection exists?
    // Walk entire selection from start to end
    for (int row=srow; row<=erow; row++) { // walk rows of selection
        const Utf8Char *u8c = u8c_ring_row(row); // ptr to first character in row
        int col_start = (row==srow) ? scol : 0; // start row? start at scol
        int col_end = (row==erow) ? ecol : ring_cols(); // end row? end at ecol
        u8c += col_start; // include col offset (if any)
        for (int col=col_start; col<=col_end; col++,u8c++) { // walk columns
            ..do something with each char at *u8c..
        }
    }
}
```

Returns:

- true – valid selection values returned
- false – no selection was made, returned values undefined

See also

`walk_selection()`, `is_selection()`

**Parameters**

<table>
<thead>
<tr>
<th>srow</th>
<th>starting row for selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>scol</td>
<td>starting column for selection</td>
</tr>
<tr>
<td>erow</td>
<td>ending row for selection</td>
</tr>
<tr>
<td>ecol</td>
<td>ending column for selection</td>
</tr>
</tbody>
</table>

**h_to_row()**

```c
int Fl_Terminal::h_to_row ( int H ) const [protected]
```

Given a height in pixels, return number of rows that “fits” into that area. This is used by the constructor to size the row/cols to fit the widget size.

**handle()**

```c
int Fl_Terminal::handle ( int e ) [virtual]
```

Handle FLTK events.
Reimplemented from Fl_Group.

**handle_unknown_char()**

```c
int Fl_Terminal::handle_unknown_char ( void ) [protected]
```

Handle an unknown char by either emitting an error symbol to the tty, or do nothing, depending on the user configurable value of `show_unknown()`.

Returns 1 if tty modified, 0 if not.
See also

    show_unknown()

12.141.5.31  history_lines()

    void Fl_Terminal::history_lines ( int val )
Set the number of lines of screen history.
Large values can be briefly heavy on cpu and memory usage.

12.141.5.32  history_use()

    int Fl_Terminal::history_use ( void ) const
Returns how many lines are "in use" by the screen history buffer.
This value will be 0 if history was recently cleared with e.g. clear_history() or "<ESC>c".
Return value will be in the range 0 .. (history_lines()-1).

12.141.5.33  insert_char()

    void Fl_Terminal::insert_char ( char c, int rep ) [protected]
Insert char ‘c’ at the current cursor position for ‘rep’ times.
Does not wrap; characters at end of line are lost.

12.141.5.34  insert_rows()

    void Fl_Terminal::insert_rows ( int count ) [protected]
Insert (count) rows at current cursor position.
Causes rows below to scroll down, and empty lines created. Lines deleted by scroll down are NOT moved into the
scroll history.

12.141.5.35  is_inside_selection()

    bool Fl_Terminal::is_inside_selection ( int grow, int gcol ) const [protected]
Is global row/column inside the current mouse selection?

Returns

    • true – (grow,gcol) is inside a valid selection.
    • false – (grow,gcol) is outside, or no valid selection.

12.141.5.36  output_translate()

    void Fl_Terminal::output_translate ( Fl_Terminal::OutFlags val )
Sets the combined output translation flags to val.
val can be sensible combinations of the OutFlags bit flags.
The default is LF_TO_CRLF, so that ‘n’ will generate both carriage-return (CR) and line-feed (LF).
For ‘r’ and ‘n’ to be handled literally, use output_translate(Fl_Terminal::OutFlags::OFF);
To disable all output translations, use 0 or Fl_Terminal::OutFlags::OFF.
12.141.5.37 print_char() [1/2]

void Fl_Terminal::print_char (char c)
Prints single ASCII char c at current cursor position, and advances the cursor.
The character is displayed at the current cursor position using the current text color/attributes.
  • c must be ASCII, not utf-8
  • Does not trigger redraws

12.141.5.38 print_char() [2/2]

void Fl_Terminal::print_char (const char *text, int len = -1)
Prints single UTF-8 char text of optional byte length len at current cursor position, and advances the cursor if
the character is printable.
Handles ASCII and control codes (CR, LF, etc).
The character is displayed at the current cursor position using the current text color/attributes.
Handles control codes and can be used to construct ANSI/XTERM escape sequences.
  • If optional len isn't specified or <0, strlen(text) is used.
  • text must not be NULL.
  • len must not be 0.
  • text must be a single char only (whether UTF-8 or ASCII)
  • text can be an ASCII character, though not as efficient as print_char()
  • Invalid UTF-8 chars show the error character (¿) depending on show_unknown(bool).
  • Does not trigger redraws

See also
  show_unknown(bool), handle_unknown_char()

12.141.5.39 printf()

void Fl_Terminal::printf (const char *fmt, ...)
Appends printf formatted messages to the terminal.
The string can contain UTF-8, crlf's, and ANSI sequences are also supported. Example:

```c
#include <FL/Fl_Terminal.H>
int main(..) {
  // Create a terminal, and append some messages to it
  Fl_Terminal *tty = new Fl_Terminal(..);
  // Append three lines of formatted text to the buffer
  tty->printf("The current date is: %s.\nThe time is: %s\n", date_str, time_str);
  tty->printf("The current PID is %ld.\n", (long) getpid());
}
```

Note
  The expanded string is currently limited to 1024 characters (including NULL). For longer strings use append() which has no string limits.

Generated by Doxygen
12.141.5.40  putchar() [1/2]

void Fl_Terminal::putchar (char c, int drow, int dcol)

Print the ASCII character c at the terminal's display position (drow,dcol). The character MUST be printable (in range 0x20 - 0x7e), and is displayed using the current text color/attributes. Characters outside that range are either ignored or print the error character (¿), depending on show_unknown(bool). This is a very low level method.

No range checking is done on drow,dcol:
  • drow must be in range 0..(display_rows()-1)
  • dcol must be in range 0..(display_columns()-1)
  • Does not trigger redraws
  • Does NOT handle control codes, ANSI or XTERM escape sequences.

See also
  show_unknown(bool), handle_unknown_char(), is_printable()

12.141.5.41  putchar() [2/2]

void Fl_Terminal::putchar (const char *text, int len, int drow, int dcol)

Print UTF-8 character text of length len at display position (drow,dcol). The character is displayed using the current text color/attributes. This is a very low level method.

No range checking is done on drow,dcol:
  • drow must be in range 0..(display_rows()-1)
  • dcol must be in range 0..(display_columns()-1)
  • Does not trigger redraws
  • Does not handle ANSI or XTERM escape sequences
  • Invalid UTF-8 chars show the error character (¿) depending on show_unknown(bool).

See also
  handle_unknown_char()

12.141.5.42  redraw_rate()

void Fl_Terminal::redraw_rate (float val)

Set the maximum rate redraw speed in floating point seconds if redraw_style() is set to RATELIMITED. When output is sent to the terminal, rather than calling redraw() right away, a timer is started with this value indicating how long to wait before calling redraw(), causing the output to be shown. 0.10 is recommended (1/10th of a second), to limit redraws to no more than 10 redraws per second.

The value that works best depends on how fast data arrives, and how fast the font system can draw text at runtime. Values too small cause too many redraws to occur, causing the terminal to get backlogged if large bursts of data arrive quickly. Values too large cause realtime output to be too "choppy".
12.141.5.43  **redraw_style()** [1/2]

```
void Fl_Terminal::redraw_style (RedrawStyle val)
```

Set how Fl_Terminal manages screen redrawing.

This setting is relevant when Fl_Terminal is used for high bandwidth data; too many redraws will slow things down, too few cause redraws to be ‘choppy’ when realtime data comes in.

Redrawing can be cpu intensive, depending on how many rows/cols are being displayed; worst case: large display + small font. Speed largely depends on the end user's graphics hardware and font drawing system.

<table>
<thead>
<tr>
<th>RedrawStyle enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_REDRAW</td>
<td>App must call <code>redraw()</code> as needed to update text to screen</td>
</tr>
<tr>
<td>RATE_LIMITED</td>
<td>Rate limited, timer controlled redraws. (DEFAULT) See <code>redraw_rate()</code></td>
</tr>
<tr>
<td>PER_WRITE</td>
<td>Redraw triggered every call to <code>append()</code> / <code>printf()</code> / etc.</td>
</tr>
</tbody>
</table>

The default style is RATE_LIMITED, which is the easiest to use, and automates redrawing to be capped at 10 redraws per second max. See `redraw_rate(float)` to control this automated redraw speed.

See also

`redraw_rate()`, `RedrawStyle`

12.141.5.44  **redraw_style()** [2/2]

```
Fl_Terminal::RedrawStyle Fl_Terminal::redraw_style () const
```

Get the redraw style.

This determines when the terminal redraws itself while text is being added to it.

Value will be one of the Fl_Terminal::RedrawStyle enum values.

See also

`redraw_style(Fl_Terminal::RedrawStyle)`

12.141.5.45  **reset_terminal()**

```
void Fl_Terminal::reset_terminal ()
```

Resets terminal to default colors, clears screen, history and mouse selection, homes cursor, resets tabstops.

Same as "<ESC>c"

12.141.5.46  **resize()**

```
void Fl_Terminal::resize (int X, int Y, int W, int H) [virtual]
```

Handle widget resizing, such as if user resizes parent window.

This may increase the column width of the widget if the width of the widget is made larger than it was.

Note

Resizing currently does not rewrap existing text. Currently enlarging makes room for longer lines, and shrinking the size lets long lines run off the right edge of the display, hidden from view. This behavior may change in the future to rewrap.

Reimplemented from Fl_Group.
12.141.5.47  scroll()

void Fl_Terminal::scroll (  
   int rows ) [protected]
Scroll the display up(+) or down(-) the specified rows.
   • Negative value scrolls "down", clearing top line, and history unaffected.
   • Postive value scrolls "up", clearing bottom line, rotating top line into history.

12.141.5.48  scrollbar_size() [1/2]

void Fl_Terminal::scrollbar_size (  
   int val )
Set the width of the scrollbar's trough to val, in pixels.
Only use this method if you need to override the global scrollbar size.
Setting val to the special value 0 causes the widget to track the global Fl::scrollbar_size().
See also
   Fl::scrollbar_size()

12.141.5.49  scrollbar_size() [2/2]

int Fl_Terminal::scrollbar_size (  
   void ) const
Get the current size of the scrollbar's trough, in pixels.
If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.
Returns
   Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.
See also
   Fl::scrollbar_size(int)

12.141.5.50  selection_extend()

bool Fl_Terminal::selection_extend (  
   int X,  
   int Y ) [protected]
Extend selection to FLTK coords X,Y.
Returns true if extended, false if nothing done (X,Y offscreen)

12.141.5.51  selection_text()

const char * Fl_Terminal::selection_text (  
   void ) const [protected]
Return text selection (for copy()/paste() operations)
   • Returns allocated NULL terminated string for entire selection.
   • Caller must free() this memory when done.
   • Unicode safe.
12.141.5.52  selection_text_len()

int Fl_Terminal::selection_text_len (  
  void ) const [protected]  
Return byte length of all UTF-8 chars in selection, or 0 if no selection.  
NOTE: Length includes trailing white on each line.

12.141.5.53  show_unknown() [1/2]

void Fl_Terminal::show_unknown (  
  bool val  )  
Set the "show unknown" flag.  
If true, unknown escape sequences and unprintable control characters will be shown with the error character "¿".  
If false, those sequences and characters will be ignored.  
See also  
  handle_unknown_char()  

12.141.5.54  show_unknown() [2/2]

bool Fl_Terminal::show_unknown (  
  void ) const  
Return the "show unknown" flag.  
See show_unknown(bool) for more info.

12.141.5.55  textattrib() [1/2]

uchar Fl_Terminal::textattrib ( ) const  
Get text attribute bits (underline, inverse, etc).  
This is the default attribute used for all newly printed text.  
See also  
  textattrib(uchar), Fl_Terminal::Attrib

12.141.5.56  textattrib() [2/2]

void Fl_Terminal::textattrib (  
  uchar val  )  
Set text attribute bits (underline, inverse, etc).  
This will be the default attribute used for all newly printed text.  
See also  
  Fl_Terminal::Attrib

12.141.5.57  textbgcolor()

void Fl_Terminal::textbgcolor (  
  Fl_Color val  )  
Set text background color to fltk color val used by any new text added.  
Use this for temporary color changes, similar to <ESC>[48;2;<R>;<G>;<B>m  
This setting does not affect the 'default' text colors used by <ESC>[0m, <ESC>c, reset_terminal(), etc. To set that too, also set textbgcolor_default(Fl_Color), e.g.
The special color value 0xffffffff (all ff's) is the "see through" color, which lets the widget's own Fl_Group::color() show through behind the text. This special text background color is the default, and is what most situations need.

See also

     textbgcolor_default(Fl_Color)

12.141.5.58  textbgcolor_default() [1/2]

void Fl_Terminal::textbgcolor_default ( Fl_Color val )

Set the default text background color used by any new text added after a reset (\texttt{\textasciitilde}<ESC>c, \texttt{\textasciitilde}<ESC>[0m, or reset_terminal()). Does not affect the 'current' text background color; use textbgcolor(Fl_Color) to set that.

The special color value 0xffffffff (all ff's) is the "see through" color, which lets the widget's own Fl_Group::color() show through behind the text. This special text background color is the default, and is what most situations need.

See also

     textbgcolor(Fl_Color)

12.141.5.59  textbgcolor_default() [2/2]

Fl_Color Fl_Terminal::textbgcolor_default ( void ) const [inline]

Return text's default background color.

See also

     textbgcolor()

12.141.5.60  textbgcolor_xterm()  

void Fl_Terminal::textbgcolor_xterm ( uchar val )

Sets the background text color as one of the 8 'xterm color' values.

This will be the foreground color used for all newly printed text, similar to the \texttt{\textasciitilde}<ESC>\{#m escape sequence, where # is between 40 and 47. This color will be reset to the default bg color if reset_terminal() is called, or by \texttt{\textasciitilde}<ESC>c, \texttt{\textasciitilde}<ESC>[0m, etc. The xterm color intensity values can be influenced by the Dim/Bold/Normal modes (which can be set with e.g. \texttt{\textasciitilde}<ESC>\{1m, textattrib(), etc), so the actual RGB values of these colors allow room for Dim/Bold to influence their brightness. For instance, "Normal Red" is not full brightness to allow "Bold Red" to be brighter. This goes for all colors except 'Black', which is not influenced by Dim or Bold; Black is always Black. The 8 color xterm values are:

- 0 = Black
- 1 = Red
- 2 = Green
- 3 = Yellow
- 4 = Blue
- 5 = Magenta
- 6 = Cyan
- 7 = White
See also
textbgcolor_default(Fl_Color)

12.141.5.61  textcolor()

void Fl_Terminal::textcolor (  
Fl_Color val  )

Set the text color for the terminal.  
This is a convenience method that sets both textfgcolor() and textfgcolor_default(), ensuring both are set to the same value.

12.141.5.62  textfgcolor()

void Fl_Terminal::textfgcolor (  
Fl_Color val  )

Set text foreground drawing color to fltk color val used by any new text added.  
Use this for temporary color changes, similar to <ESC>[38;2;<R>;<G>;<B>m.  
This setting does not affect the 'default' text colors used by <ESC>[0m, <ESC>c, reset_terminal(), etc. To change both the current and default fg color, also use textfgcolor_default(Fl_Color). Example:

// Set both 'current' and 'default' colors  
Fl_Color amber = 0xd0704000;  
tty->textfgcolor(amber);  // set 'current' fg color  
tty->textfgcolor_default(amber);  // set 'default' fg color used by ESC[0m reset

See also
textfgcolor_default(Fl_Color)

12.141.5.63  textfgcolor_default() [1/2]

void Fl_Terminal::textfgcolor_default (  
Fl_Color val  )

Set the default text foreground color used by <ESC>c, <ESC>[0m, and reset_terminal().  
Does not affect the 'current' text foreground color; use textfgcolor(Fl_Color) to set that.

See also
textfgcolor(Fl_Color)

12.141.5.64  textfgcolor_default() [2/2]

Fl_Color Fl_Terminal::textfgcolor_default (  
void ) const [inline]

Return text's default foreground color.

See also
textfgcolor()

12.141.5.65  textfgcolor_xterm()

void Fl_Terminal::textfgcolor_xterm (  
uchar val  )

Sets the foreground text color as one of the 8 'xterm color' values.  
This will be the foreground color used for all newly printed text, similar to the <ESC>[#m escape sequence, where # is between 30 and 37.  
This color will be reset to the default fg color if reset_terminal() is called, or by <ESC>c, <ESC>[0m, etc.
The xterm color intensity values can be influenced by the Dim/Bold/Normal modes (which can be set with e.g. 
<ESC>[1m, textattrib(), etc), so the actual RGB values of these colors allow room for Dim/Bold to influence their 
brightness. For instance, "Normal Red" is not full brightness to allow "Bold Red" to be brighter. This goes for all 
colors except 'Black', which is not influenced by Dim or Bold; Black is always Black.
The 8 color xterm values are:

- 0 = Black
- 1 = Red
- 2 = Green
- 3 = Yellow
- 4 = Blue
- 5 = Magenta
- 6 = Cyan
- 7 = White

See also

textfgcolor_default(Fl_Color)

12.141.5.66  textfont()

void Fl_Terminal::textfont ( 
    Fl_Font val )

Sets the font used for all text displayed in the terminal.
This affects all existing text (in display and history) as well as any newly printed text.
Only monospace fonts are recommended, such as FL_COURIER or FL_SCREEN. Custom fonts configured with 
Fl::set_font() will also work, as long as they are monospace.

12.141.5.67  textsize()

void Fl_Terminal::textsize ( 
    Fl_Fontsize val )

Sets the font size used for all text displayed in the terminal.
This affects all existing text (in display and history) as well as any newly printed text.
Changing this will affect the display_rows() and display_columns().

12.141.5.68  u8cDisp_row()

Fl_Terminal::Utf8Char * Fl_Terminal::u8cDisp_row ( 
    int drow ) [protected]

Return pointer to the first u8c character in row drow of the display.

- 'drow' is indexed relative to the beginning of the display buffer.

- This can be used to walk all columns in the specified row, e.g.
  
  `// Print all chars in first row of display (ASCII and UTF-8)`
  `Utf8Char u8c = u8cDisp_row(0);`  // first char of first display row
  `int scol = 0, ecol = disp_cols();`  // start/end for column loop
  `for (int col=scol; col<ecol; col++, u8c++) {`  // loop from first char to last
      `char *text = u8c->text_utf8();`  // text string for char
      `int len = u8c->length();`  // text string length for char
      `::printf("<%.*s>", len, text);`  // print potentially multibyte char
  }`
12.141.5.69  u8c_hist_row()

Fl_Terminal::Utf8Char * Fl_Terminal::u8c_hist_row (  
    int hrow ) [protected]

Return u8c for beginning of a row inside the scrollback history.  
'hrow' is indexed relative to the beginning of the scrollback history buffer.

See also

    u8c_disp_row(int) for example use.

12.141.5.70  u8c_hist_use_row()

Fl_Terminal::Utf8Char * Fl_Terminal::u8c_hist_use_row (  
    int hurow ) [protected]

Return u8c for beginning of row hurow inside the 'in use' part of the scrollback history.  
'hurow' is indexed relative to the beginning of the 'in use' part of the scrollback history buffer. This may be a different from u8c_hist_row(int) if the history was recently cleared, and there aren't many (or any) rows in the history buffer that have been populated with scrollback text yet.

See also

    u8c_disp_row(int) for example use.

12.141.5.71  u8c_ring_row()

Fl_Terminal::Utf8Char * Fl_Terminal::u8c_ring_row (  
    int grow ) [protected]

Return UTF-8 char for row grow in the ring buffer.  
grow is globally indexed relative to the beginning of the ring buffer, so this method can access ANY character in the entire ring buffer (hist or disp) by the global index.
Scolling offset is NOT applied; this is raw access to the ring's rows.  Should really ONLY be used for making a complete copy of the ring.

Example:
// Walk ALL rows and cols in the ring buffer..
for (int row=0; row<ring.rows(); row++) {  
    Utf8Char *u8c = ring.u8c_ring_row(row);  
    for (int col=0; col<ring.cols(); col++,u8c++) {  
        ..make use of u8c->xxx() methods..  
    }  
}

12.141.5.72  vprintf()

void Fl_Terminal::vprintf (  
    const char * fmt,  
    va_list ap )

Appends printf formatted messages to the terminal.  
Subclasses can use this to implement their own printf() functionality.  
The string can contain UTF-8, crlf's, and ANSI sequences are also supported when ansi(bool) is set to 'true'.

Note
The expanded string is currently limited to 1024 characters (including NULL). For longer strings use append() which has no string limits.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmt</td>
<td>is a printf format string for the message text.</td>
</tr>
<tr>
<td>ap</td>
<td>is a va_list created by va_start() and closed with va_end(), which the caller is responsible for handling.</td>
</tr>
</tbody>
</table>
12.141.5.73  \textbf{w\_to\_col()}

\begin{verbatim}
int Fl_Terminal::w_to_col (int W) const [protected]
\end{verbatim}

Given a width in pixels, return number of columns that "fits" into that area. This is used by the constructor to size the row/cols to fit the widget size.

12.141.5.74  \textbf{walk\_selection()}

\begin{verbatim}
const Fl_Terminal::Utf8Char * Fl_Terminal::walk_selection (const Utf8Char * u8c,
int & row,
int & col) const [protected]
\end{verbatim}

Walk the mouse selection one character at a time from beginning to end, returning a Utf8Char* to the next character in the selection, or NULL if the end was reached, or if there's no selection. This is easier to use for walking the selection than \texttt{get\_selection()}. \texttt{u8c} should start out as NULL, rewinding to the beginning of the selection. If the returned Utf8Char* is not NULL, \texttt{row} and \texttt{col} return the character's row/column position in the ring buffer.

\begin{verbatim}
// EXAMPLE: Walk the entire mouse selection, if any
int row,col; // the returned row/col for each char
Utf8Char *u8c = NULL; // start with NULL to begin walk
while ((u8c = walk_selection(u8c, row, col))) { // loop until end char reached
   .do something with *u8c..
}
\end{verbatim}

See also \texttt{get\_selection()}, \texttt{is\_selection()}

\begin{tabular}{|c|}
\hline
\texttt{u8c} & NULL on first iter \\
\hline
\texttt{row} & returned row# \\
\hline
\texttt{col} & returned col# \\
\hline
\end{tabular}

12.141.6  \textbf{Member Data Documentation}

12.141.6.1  \textbf{scrollbar}

\begin{verbatim}
Fl_Scrollbar* Fl_Terminal::scrollbar
\end{verbatim}

Vertical scrollbar. Public, so that it can be accessed directly.

\textbf{Todo} Support scrollbar\_left/right() - See \texttt{Fl\_Browser\_::scrollbar} docs

The documentation for this class was generated from the following files:

\begin{itemize}
\item \texttt{Fl\_Terminal.H}
\item \texttt{Fl\_Terminal.cxx}
\end{itemize}

12.142  \textbf{Fl\_Text\_Buffer Class Reference}

This class manages Unicode text displayed in one or more \texttt{Fl\_Text\_Display} widgets.

\texttt{#include <Fl\_Text\_Buffer.H>}

\begin{itemize}
\item \texttt{void add\_modify\_callback (Fl\_Text\_Modify\_Cb bufModifiedCB, void *cbArg)}
\end{itemize}
12.142 Fl_Text_Buffer Class Reference

Adds a callback function that is called whenever the text buffer is modified.

- void **add_predelete_callback** (Fl_Text_Predelete_Cb bufPredelCB, void *cbArg)

  Adds a callback routine to be called before text is deleted from the buffer.

- void **add_modify_callback** (void *cbArg)

  Calls all modify callbacks that have been registered using the add_modify_callback() method.

- void **add_predelete_callbacks** ()

  Calls the stored pre-delete callback procedure(s) for this buffer to update the changed area(s) on the screen and any other listeners.

- bool **can_redo** () const

  Check if undo is enabled and if the last undo action can be redone.

- bool **can_undo** () const

  Check if undo is enabled and if the last action can be undone.

- void **canUndo** (char flag=1)

  Enable or disable undo actions for this text buffer.

- unsigned int **char_at** (int pos) const

  Returns the character at the specified position pos in the buffer.

- void **copy** (Fl_Text_Buffer *fromBuf, int fromStart, int fromEnd, int toPos)

  Copies text from another Fl_Text_Buffer to this one.

- int **count_displayed_characters** (int lineStartPos, int targetPos) const

  Count the number of displayed characters between buffer position lineStartPos and targetPos.

- int **count_lines** (int startPos, int endPos) const

  Counts the number of newlines between startPos and endPos in buffer.

- int **findchar_backward** (int startPos, unsigned int searchChar, int *foundPos) const

  Search backwards in buffer buf for character searchChar, starting with the character before startPos, returning the result in foundPos.

- int **findchar_forward** (int startPos, unsigned searchChar, int *foundPos) const

  Finds the next occurrence of the specified character.

- Fl_Text_Buffer (int requestedSize=0, int preferredGapSize=1024)

  Create an empty text buffer of a pre-determined size.

- int **highlight** ()

  Returns a non-zero value if text has been highlighted, 0 otherwise.

- void **highlight** (int start, int end)

  Highlights the specified text within the buffer.

- int **highlight_position** (int *start, int *end)

  Highlights the specified text between start and end within the buffer.

- const Fl_Text_Selection * **highlight_selection** () const

  Returns the current highlight selection.

- char * **highlight_text** ()

  Returns the highlighted text.

- void **insert** (int pos, const char *text, int insertedLength=-1)

  Inserts null-terminated string text at position pos.
• int insertfile (const char *file, int pos, int buflen=128 *1024)
  Inserts a file at the specified position.
• bool is_word_separator (int pos) const
  Returns whether character at position pos is a word separator.
• int length () const
  Returns the number of bytes in the buffer.
• int line_end (int pos) const
  Finds and returns the position of the end of the line containing position pos (which is either a pointer to the newline character ending the line or a pointer to one character beyond the end of the buffer).
• int line_start (int pos) const
  Returns the position of the start of the line containing position pos.
• char * line_text (int pos) const
  Returns the text from the entire line containing the specified character position.
• int loadfile (const char *file, int buflen=128 *1024)
  Loads a text file into the buffer.
• int next_char (int ix) const
  Returns the index of the next character.
• int next_char_clipped (int ix) const
• int outputfile (const char *file, int start, int end, int buflen=128 *1024)
  Writes the specified portions of the text buffer to a file.
• int prev_char (int ix) const
  Returns the index of the previous character.
• int prev_char_clipped (int ix) const
• Fl_Text_Selection * primary_selection ()
  Returns the primary selection.
• const Fl_Text_Selection * primary_selection () const
  Returns the primary selection.
• void printf (const char *fmt,...)
  Appends printf formatted messages to the end of the buffer.
• int redo (int *cp=0)
  Redo previous undo action.
• void remove (int start, int end)
  Deletes a range of characters in the buffer.
• void remove_modify_callback (Fl_Text_Modify_Cb bufModifiedCB, void *cbArg)
  Removes a modify callback.
• void remove_predelete_callback (Fl_Text_Predelete_Cb predelCB, void *cbArg)
  Removes a callback routine bufPreDeleteCB associated with argument cbArg to be called before text is deleted from the buffer.
• void remove_secondary_selection ()
  Removes the text from the buffer corresponding to the secondary text selection object.
• void remove_selection ()
  Removes the text in the primary selection.
• void replace (int start, int end, const char *text, int insertedLength=-1)
  Deletes the characters between start and end, and inserts the null-terminated string text in their place in the buffer.
• void replace_secondary_selection (const char *text)
  Replaces the text from the buffer corresponding to the secondary text selection object with the new string text.
• void replace_selection (const char *text)
  Replaces the text in the primary selection.
• int rewind_lines (int startPos, int nLines)
  Finds and returns the position of the first character of the line nLines backwards from startPos (not counting the character pointed to by startPos if that is a newline) in the buffer.
- int `savefile` (const char *file, int buflen=128 * 1024)
  Saves a text file from the current buffer.
- int `search_backward` (int startPos, const char *searchString, int *foundPos, int matchCase=0) const
  Search backwards in buffer for string `searchString`, starting with the character at `startPos`, returning the result in `foundPos`.
- int `search_forward` (int startPos, const char *searchString, int *foundPos, int matchCase=0) const
  Search forwards in buffer for string `searchString`, starting with the character `startPos`, and returning the result in `foundPos`.
- void `secondary_select` (int start, int end)
  Selects a range of characters in the secondary selection.
- int `secondary_selected` () const
  Returns a non-zero value if text has been selected in the secondary text selection, 0 otherwise.
- const Fl_Text_Selection * `secondary_selection` () const
  Returns the secondary selection.
- int `secondary_selection_position` (int *start, int *end) const
  Returns the current selection in the secondary text selection object.
- char * `secondary_selection_text` ()
  Returns the text in the secondary selection.
- void `secondary_unselect` ()
  Clears any selection in the secondary text selection object.
- void `select` (int start, int end)
  Selects a range of characters in the buffer.
- int `selected` () const
  Returns a non-zero value if text has been selected, 0 otherwise.
- int `selection_position` (int *start, int *end) const
  Gets the selection position.
- char * `selection_text` ()
  Returns the currently selected text.
- int `skip_displayed_characters` (int lineStartPos, int nChars)
  Count forward from buffer position `lineStartPos` in displayed characters.
- int `skip_lines` (int startPos, int nLines)
  Finds the first character of the line `nLines` forward from `startPos` in the buffer and returns its position.
- int `tab_distance` () const
  Gets the tab width.
- void `tab_distance` (int tabDist)
  Set the hardware tab distance (width) used by all displays for this buffer, and used in computing offsets for rectangular selection operations.
- char * `text` () const
  Get a copy of the entire contents of the text buffer.
- void `text` (const char *text)
  Replaces the entire contents of the text buffer.
- char * `text_range` (int start, int end) const
  Get a copy of a part of the text buffer.
- int `undo` (int *cp=0)
  Undo text modification according to the undo variables or insert text from the undo buffer.
- void `unhighlight` ()
  Unhighlights text in the buffer.
- void `unselect` ()
  Cancels any previous selection on the primary text selection object.
- int `utf8_align` (int) const
  Align an index into the buffer to the current or previous UTF-8 boundary.
• void vprintf (const char *fmt, va_list ap)
  Can be used by subclasses that need their own printf() style functionality.

• int word_end (int pos) const
  Returns the position corresponding to the end of the word.

• int word_start (int pos) const
  Returns the position corresponding to the start of the word.

• ~Fl_Text_Buffer ()
  Frees a text buffer.

Public Attributes

• int input_file_was_transcoded
  true if the loaded file has been transcoded to UTF-8.

• void(transcoding_warning_action)(Fl_Text_Buffer *)
  Pointer to a function called after reading a non UTF-8 encoded file.

Static Public Attributes

• static const char * file_encoding_warning_message
  This message may be displayed using the fl_alert() function when a file which was not UTF-8 encoded is input.

Protected Member Functions

• int apply_undo (Fl_Text_Undo_Action *action, int *cursorPos)
  Apply the current undo/redo operation, called from undo() or redo().

• void call_modify_callbacks (int pos, int nDeleted, int nInserted, int nRestyled, const char *deletedText) const
  Calls the stored modify callback procedure(s) for this buffer to update the changed area(s) on the screen and any other listeners.

• void call_predelete_callbacks (int pos, int nDeleted) const
  Calls the stored pre-delete callback procedure(s) for this buffer to update the changed area(s) on the screen and any other listeners.

• int insert_ (int pos, const char *text, int insertedLength=-1)
  Internal (non-redisplaying) version of insert().

• void move_gap (int pos)
  Move the gap to start at a new position.

• void reallocate_with_gap (int newGapStart, int newGapLen)
  Reallocates the text storage in the buffer to have a gap starting at newGapStart and a gap size of newGapLen, preserving the buffer’s current contents.

• void redisplay_selection (Fl_Text_Selection *oldSelection, Fl_Text_Selection *newSelection) const
  Calls the stored redisplay procedure(s) for this buffer to update the screen for a change in a selection.

• void remove_ (int start, int end)
  Internal (non-redisplaying) version of remove().

• void remove_selection_ (Fl_Text_Selection *sel)
  Removes the text from the buffer corresponding to sel.

• void replace_selection_ (Fl_Text_Selection *sel, const char *text)
  Replaces the text in selection sel.

• char * selection_text_ (Fl_Text_Selection *sel) const

• void update_selections (int pos, int nDeleted, int nInserted)
  Updates all of the selections in the buffer for changes in the buffer’s text.
Protected Attributes

- char * mBuf
  allocated memory where the text is stored
- char mCanUndo
  if this buffer is used for attributes, it must not do any undo calls
- void ** mCbArgs
  caller arguments for modifyProcs above
- int mCursorPosHint
  hint for reasonable cursor position after a buffer modification operation
- int mGapEnd
  points to the first character after the gap
- int mGapStart
  points to the first character of the gap
- Fl_Text_Selection mHighlight
  highlighted areas
- int mLength
  length of the text in the buffer (the length of the buffer itself must be calculated: gapEnd - gapStart + length)
- Fl_Text_Modify_Cb * mModifyProcs
  procedures to call when buffer is modified to redisplay contents
- int mNModifyProcs
  number of modify-redisplay procs attached
- int mNPredelProcs
  number of pre-delete procs attached
- void ** mPredeleteCbArgs
  caller argument for pre-delete proc above
- Fl_Text_Predel_Cb * mPredelProcs
  procedure to call before text is deleted from the buffer; at most one is supported.
- int mPreferredGapSize
  the default allocation for the text gap is 1024 bytes and should only be increased if frequent and large changes in buffer size are expected
- Fl_Text_Selection mPrimary
  highlighted areas
- Fl_Text_Undo_Action_List * mRedoList
  List of redo event.
- Fl_Text_Selection mSecondary
  highlighted areas
- int mTabDist
  equiv.
- Fl_Text_Undo_Action * mUndo
  local undo event
- Fl_Text_Undo_Action_List * mUndoList
  List of undo event.

12.142.1 Detailed Description

This class manages Unicode text displayed in one or more Fl_Text_Display widgets. All text in Fl_Text_Buffer must be encoded in UTF-8. All indices used in the function calls must be aligned to the start of a UTF-8 sequence. All indices and pointers returned will be aligned. All functions that return a single character will return that in an unsigned int in UCS-4 encoding. The Fl_Text_Buffer class is used by the Fl_Text_Display and Fl_Text_Editor to manage complex text data and is based upon the excellent NEdit text editor engine - see https://sourceforge.net/projects/nedit/.
12.142.2 Constructor & Destructor Documentation

12.142.2.1 Fl_Text_Buffer()

Fl_Text_Buffer::Fl_Text_Buffer (  
    int requestedSize = 0,  
    int preferredGapSize = 1024  
)  
Create an empty text buffer of a pre-determined size.

Parameters

<table>
<thead>
<tr>
<th>requestedSize</th>
<th>use this to avoid unnecessary re-allocation if you know exactly how much the buffer will need to hold</th>
</tr>
</thead>
<tbody>
<tr>
<td>preferredGapSize</td>
<td>Initial size for the buffer gap (empty space in the buffer where text might be inserted if the user is typing sequential characters)</td>
</tr>
</tbody>
</table>

12.142.3 Member Function Documentation

12.142.3.1 add_modify_callback()

void Fl_Text_Buffer::add_modify_callback (  
    Fl_Text_Modify_Cb bufModifiedCB,  
    void * cbArg  
)  
Adds a callback function that is called whenever the text buffer is modified. The callback function is declared as follows:

typedef void (*Fl_Text_Modify_Cb)(int pos, int nInserted, int nDeleted,  
                                 int nRestyled, const char* deletedText,  
                                 void* cbArg);

12.142.3.2 address() [1/2]

char * Fl_Text_Buffer::address (  
    int pos  
)  [inline]  
Convert a byte offset in buffer into a memory address.

Parameters

| pos | byte offset into buffer |

Returns

byte offset converted to a memory address

12.142.3.3 address() [2/2]

const char * Fl_Text_Buffer::address (  
    int pos  
) const  [inline]  
Convert a byte offset in buffer into a memory address.

Parameters

| pos | byte offset into buffer |

Returns

byte offset converted to a memory address
12.142.3.4  append()

void Fl_Text_Buffer::append (  
    const char * t,  
    int addedLength = -1 ) [inline]

Appends the text string to the end of the buffer.

Parameters

<table>
<thead>
<tr>
<th>t</th>
<th>UTF-8 encoded text</th>
</tr>
</thead>
<tbody>
<tr>
<td>addedLength</td>
<td>number of bytes to append, or -1 to indicate t is null-terminated</td>
</tr>
</tbody>
</table>

12.142.3.5  appendfile()

int Fl_Text_Buffer::appendfile (  
    const char * file,  
    int buflen = 128*1024 ) [inline]

Appends the named file to the end of the buffer.
See also insertfile().

12.142.3.6  byte_at()

char Fl_Text_Buffer::byte_at (  
    int pos ) const

Returns the raw byte at the specified position pos in the buffer.
Positions start at 0.

Parameters

| pos  | byte offset into buffer |

Returns

unencoded raw byte

12.142.3.7  can_redo()

bool Fl_Text_Buffer::can_redo ( ) const

Check if undo is enabled and if the last undo action can be redone.

Check if undo is enabled and if the last undo action can be redone.
See also
canUndo()

12.142.3.8  can_undo()

bool Fl_Text_Buffer::can_undo ( ) const

Check if undo is enabled and if the last action can be undone.

See also
canUndo()

12.142.3.9  canUndo()

void Fl_Text_Buffer::canUndo (  
    char flag = 1 )
Enable or disable undo actions for this text buffer.
Undo actions are enabled for text buffer by default. If used as a style buffer in Fl_Text_Display, undo actions are
disabled as they are handled by the text buffer.

See also
    can_undo()

12.142.3.10  char_at()

    unsigned int Fl_Text_Buffer::char_at ( int pos ) const

Returns the character at the specified position pos in the buffer.
Positions start at 0.

Parameters

    pos  byte offset into buffer, pos must be at a UTF-8 character boundary

Returns
    Unicode UCS-4 encoded character

12.142.3.11  copy()

    void Fl_Text_Buffer::copy ( Fl_Text_Buffer ∗ fromBuf, int fromStart, int fromEnd, int toPos )

Copies text from another Fl_Text_Buffer to this one.

Parameters

    fromBuf  source text buffer, may be the same as this
    fromStart  byte offset into buffer
    fromEnd  byte offset into buffer
    toPos  destination byte offset into buffer

12.142.3.12  count_displayed_characters()

    int Fl_Text_Buffer::count_displayed_characters ( int lineStartPos, int targetPos ) const

Count the number of displayed characters between buffer position lineStartPos and targetPos.
Displayed characters are the characters shown on the screen to represent characters in the buffer, where tabs and
control characters are expanded.

12.142.3.13  count_lines()

    int Fl_Text_Buffer::count_lines ( int startPos, int endPos ) const

Counts the number of newlines between startPos and endPos in buffer.
The character at position endPos is not counted.
12.142.3.14 findchar_backward()

```cpp
int Fl_Text_Buffer::findchar_backward (  
    int startPos,  
    unsigned int searchChar,  
    int * foundPos ) const
```

Search backwards in buffer buf for character searchChar, starting with the character before startPos, returning the result in foundPos.

Returns 1 if found, 0 if not. The difference between this and search_backward() is that it's optimized for single characters. The overall performance of the text widget is dependent on its ability to count lines quickly, hence searching for a single character: newline.

**Parameters**

<table>
<thead>
<tr>
<th>startPos</th>
<th>byte offset to start position</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchChar</td>
<td>UCS-4 character that we want to find</td>
</tr>
<tr>
<td>foundPos</td>
<td>byte offset where the character was found</td>
</tr>
</tbody>
</table>

**Returns**

1 if found, 0 if not

12.142.3.15 findchar_forward()

```cpp
int Fl_Text_Buffer::findchar_forward (  
    int startPos,  
    unsigned searchChar,  
    int * foundPos ) const
```

Finds the next occurrence of the specified character. Search forwards in buffer for character searchChar, starting with the character startPos, and returning the result in foundPos. Returns 1 if found, 0 if not. The difference between this and search_forward() is that it's optimized for single characters. The overall performance of the text widget is dependent on its ability to count lines quickly, hence searching for a single character: newline.

**Parameters**

<table>
<thead>
<tr>
<th>startPos</th>
<th>byte offset to start position</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchChar</td>
<td>UCS-4 character that we want to find</td>
</tr>
<tr>
<td>foundPos</td>
<td>byte offset where the character was found</td>
</tr>
</tbody>
</table>

**Returns**

1 if found, 0 if not

12.142.3.16 highlight_text()

```cpp
char * Fl_Text_Buffer::highlight_text ( )
```

Returns the highlighted text.

When you are done with the text, free it using the free() function.

12.142.3.17 insert()

```cpp
void Fl_Text_Buffer::insert (  
    int pos,  
    const char * text,  
    int insertedLength = -1 )
```

Inserts null-terminated string text at position pos.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>insertion position as byte offset (must be UTF-8 character aligned)</td>
</tr>
<tr>
<td>text</td>
<td>UTF-8 encoded text</td>
</tr>
<tr>
<td>insertedLength</td>
<td>number of bytes to insert, or -1 to indicate text is null-terminated</td>
</tr>
</tbody>
</table>

12.142.3.18  **insert()**

```cpp
int Fl_Text_Buffer::insert_ (  
  int pos,  
  const char * text,  
  int insertedLength = -1 ) [protected]
```

Internal (non-redisplaying) version of `insert()`.

Returns the length of text inserted (this is just `strlen(text)` if `insertedLength` == `-1`, however this calculation can be expensive and the length will be required by any caller who will continue on to call redisplay). `pos` must be contiguous with the existing text in the buffer (i.e. not past the end).

Returns

the number of bytes inserted

12.142.3.19  **insertfile()**

```cpp
int Fl_Text_Buffer::insertfile (  
  const char * file,  
  int pos,  
  int buflen = 128*1024 )
```

Inserts a file at the specified position.

Returns

- 0 on success
- non-zero on error (`strerror()` contains reason)
- 1 indicates open for read failed (no data loaded)
- 2 indicates error occurred while reading data (data was partially loaded)

File can be UTF-8 or CP1252 encoded. If the input file is not UTF-8 encoded, the `Fl_Text_Buffer` widget will contain data transcoded to UTF-8. By default, the message `Fl_Text_Buffer::file_encoding_warning_message` will warn the user about this.

See also

`input_file_was_transcoded` and `transcoding_warning_action`.

12.142.3.20  **is_word_separator()**

```cpp
bool Fl_Text_Buffer::is_word_separator (  
  int pos ) const
```

Returns whether character at position `pos` is a word separator.

Pos must be at a character boundary.

12.142.3.21  **length()**

```cpp
int Fl_Text_Buffer::length ( ) const [inline]
```

Returns the number of bytes in the buffer.

Returns

size of text in bytes
12.142.3.22  line_end()

int Fl_Text_Buffer::line_end (  
    int pos ) const

Finds and returns the position of the end of the line containing position pos (which is either a pointer to the newline character ending the line or a pointer to one character beyond the end of the buffer).

Parameters

| pos | byte index into buffer |

Returns

byte offset to line end

12.142.3.23  line_start()

int Fl_Text_Buffer::line_start (  
    int pos ) const

Returns the position of the start of the line containing position pos.

Parameters

| pos | byte index into buffer |

Returns

byte offset to line start

12.142.3.24  line_text()

c char * Fl_Text_Buffer::line_text (  
    int pos ) const

Returns the text from the entire line containing the specified character position. When you are done with the text, free it using the free() function.

Parameters

| pos | byte index into buffer |

Returns

copy of UTF-8 text, must be free'd

12.142.3.25  loadfile()

int Fl_Text_Buffer::loadfile (  
    const char * file,  
    int buflen = 128*1024 ) [inline]

Loads a text file into the buffer.
See also insertfile().

12.142.3.26  next_char()

int Fl_Text_Buffer::next_char (  
    int ix ) const
Returns the index of the next character.
12.142 Fl_Text_Buffer Class Reference

Parameters

*ix*  |  index to the current character

### 12.142.3.27 outputfile()

```cpp
int Fl_Text_Buffer::outputfile (
    const char * file,
    int start,
    int end,
    int buflen = 128*1024 )
```

Writes the specified portions of the text buffer to a file.

**Returns**

- 0 on success
- non-zero on error (strerror() contains reason)
- 1 indicates open for write failed (no data saved)
- 2 indicates error occurred while writing data (data was partially saved)

**See also**

savefile(const char *file, int buflen)

### 12.142.3.28 prev_char()

```cpp
int Fl_Text_Buffer::prev_char ( int ix ) const
```

Returns the index of the previous character.

Parameters

*ix*  |  index to the current character

### 12.142.3.29 printf()

```cpp
void Fl_Text_Buffer::printf ( const char * fmt,
    ... )
```

Appends printf formatted messages to the end of the buffer.

**Example:**

```cpp
#include <FL/Fl_Text_Display.H>
int main(..) { 
    // Create a text display widget and assign it a text buffer
    Fl_Text_Display *tdsp = new Fl_Text_Display();
    Fl_Text_Buffer *tbuf = new Fl_Text_Buffer();
    tdsp->buffer(tbuf);
     // Append three lines of formatted text to the buffer
    tbuf->printf("The current date is: %s\nThe time is: %s\n", date_str, time_str);
    tbuf->printf("The current PID is %ld\n", (long)getpid());
};
```

**Note**

The expanded string is currently limited to 1024 characters.
Parameters

\textit{fmt} is a printf format string for the message text.

\subsection{remove()}

\begin{verbatim}
void Fl_Text_Buffer::remove (  
    int start,  
    int end )
\end{verbatim}

Deletes a range of characters in the buffer.

Parameters

- \textit{start}: byte offset to first character to be removed
- \textit{end}: byte offset to character after last character to be removed

\subsection{remove()}

\begin{verbatim}
void Fl_Text_Buffer::remove_ (  
    int start,  
    int end ) [protected]
\end{verbatim}

Internal (non-redisplaying) version of \textit{remove()}. Removes the contents of the buffer between \textit{start} and \textit{end} (and moves the gap to the site of the delete).

\subsection{replace()}

\begin{verbatim}
void Fl_Text_Buffer::replace (  
    int start,  
    int end,  
    const char * text,  
    int insertedLength = -1 )
\end{verbatim}

Deletes the characters between \textit{start} and \textit{end}, and inserts the null-terminated string \textit{text} in their place in the buffer.

Parameters

- \textit{start}: byte offset to first character to be removed and new insert position
- \textit{end}: byte offset to character after last character to be removed
- \textit{text}: UTF-8 encoded text
- \textit{insertedLength}: number of bytes to insert, or -1 to indicate \textit{text} is null-terminated

\subsection{rewind_lines()}

\begin{verbatim}
int Fl_Text_Buffer::rewind_lines (  
    int startPos,  
    int nLines )
\end{verbatim}

Finds and returns the position of the first character of the line \textit{nLines} backwards from \textit{startPos} (not counting the character pointed to by \textit{startpos} if that is a newline) in the buffer. \textit{nLines} == 0 means find the beginning of the line.

\subsection{savefile()}

\begin{verbatim}
int Fl_Text_Buffer::savefile (  
    const char * file,  
    int buflen = 128*1024 ) [inline]
\end{verbatim}
Saves a text file from the current buffer.

Returns

- 0 on success
- non-zero on error (strerror() contains reason)
- 1 indicates open for write failed (no data saved)
- 2 indicates error occurred while writing data (data was partially saved)

See also

outputfile(const char *file, int start, int end, int buflen)

12.142.3.35 search_backward()

int Fl_Text_Buffer::search_backward {
    int startPos,
    const char * searchString,
    int * foundPos,
    int matchCase = 0 ) const

Search backwards in buffer for string searchString, starting with the character at startPos, returning the result in foundPos.

Returns 1 if found, 0 if not.

Parameters

| startPos  | byte offset to start position |
| searchString | UTF-8 string that we want to find |
| foundPos | byte offset where the string was found |
| matchCase | if set, match character case |

Returns 1 if found, 0 if not.

12.142.3.36 search_forward()

int Fl_Text_Buffer::search_forward {
    int startPos,
    const char * searchString,
    int * foundPos,
    int matchCase = 0 ) const

Search forwards in buffer for string searchString, starting with the character startPos, and returning the result in foundPos.

Returns 1 if found, 0 if not.

Parameters

| startPos  | byte offset to start position |
| searchString | UTF-8 string that we want to find |
| foundPos | byte offset where the string was found |
| matchCase | if set, match character case |
Returns

1 if found, 0 if not

12.142.3.37 secondary_selection_text()

char * Fl_Text_Buffer::secondary_selection_text ( )
Returns the text in the secondary selection.
When you are done with the text, free it using the free() function.

12.142.3.38 selection_text()

char * Fl_Text_Buffer::selection_text ( )
Returns the currently selected text.
When you are done with the text, free it using the free() function.

12.142.3.39 skip_displayed_characters()

int Fl_Text_Buffer::skip_displayed_characters ( int lineStartPos, int nChars )
Count forward from buffer position `lineStartPos` in displayed characters.
Displayed characters are the characters shown on the screen to represent characters in the buffer, where tabs and control characters are expanded.

Parameters

<table>
<thead>
<tr>
<th>lineStartPos</th>
<th>byte offset into buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>nChars</td>
<td>number of bytes that are sent to the display</td>
</tr>
</tbody>
</table>

Returns

byte offset in input after all output bytes are sent

12.142.3.40 tab_distance()

int Fl_Text_Buffer::tab_distance ( ) const [inline]
Gets the tab width.
The tab width is measured in characters. The pixel position is calculated using an average character width.

12.142.3.41 text() [1/2]

char * Fl_Text_Buffer::text ( ) const
Get a copy of the entire contents of the text buffer.
Memory is allocated to contain the returned string, which the caller must free.

Returns

newly allocated text buffer - must be free'd, text is UTF-8

12.142.3.42 text() [2/2]

void Fl_Text_Buffer::text ( const char * text )
Replaces the entire contents of the text buffer.

Parameters

| text | Text must be valid UTF-8. If null, an empty string is substituted. |
12.142.3.43  text_range()

char * Fl_Text_Buffer::text_range (  
   int start,  
   int end ) const

Get a copy of a part of the text buffer.
Return a copy of the text between start and end character positions from text buffer buf. Positions start at 0, and the range does not include the character pointed to by end. When you are done with the text, free it using the free() function.

Parameters

<table>
<thead>
<tr>
<th>start</th>
<th>byte offset to first character</th>
</tr>
</thead>
<tbody>
<tr>
<td>end</td>
<td>byte offset after last character in range</td>
</tr>
</tbody>
</table>

Returns

newly allocated text buffer - must be free'd, text is UTF-8

12.142.3.44  undo()

int Fl_Text_Buffer::undo (  
   int * cursorPos = 0 )

Undo text modification according to the undo variables or insert text from the undo buffer.
Take the previous changes and undo them.
Return the previous cursor position in cursorPos. Returns 1 if the undo was applied. CursorPos will be at a character boundary.

12.142.3.45  vprintf()

void Fl_Text_Buffer::vprintf (  
   const char * fmt,  
   va_list ap )

Can be used by subclasses that need their own printf() style functionality.

Note

The expanded string is currently limited to 1024 characters.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fmt</th>
<th>is a printf format string for the message text.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ap</td>
<td>is a va_list created by va_start() and closed with va_end(), which the caller is responsible for handling.</td>
</tr>
</tbody>
</table>

12.142.3.46  word_end()

int Fl_Text_Buffer::word_end (  
   int pos ) const

Returns the position corresponding to the end of the word.

Parameters

| pos | byte index into buffer |

Generated by Doxygen
Returns
byte offset to word end

12.142.3.47 word_start()

```c
int Fl_Text_Buffer::word_start (int pos ) const
```

Returns the position corresponding to the start of the word.

Parameters

| pos | byte index into buffer |

Returns
byte offset to word start

### 12.142.4 Member Data Documentation

#### 12.142.4.1 file_encoding_warning_message

```c
const char * Fl_Text_Buffer::file_encoding_warning_message [static]
```

Initial value:

```
"Displayed text contains the UTF-8 transcoding\n"of the input file which was not UTF-8 encoded.\n"Some changes may have occurred."
```

This message may be displayed using the fl_alert() function when a file which was not UTF-8 encoded is input.

#### 12.142.4.2 mTabDist

```c
int Fl_Text_Buffer::mTabDist [protected]
```

equiv. number of characters in a tab

#### 12.142.4.3 transcoding_warning_action

```c
void(* Fl_Text_Buffer::transcoding_warning_action) (Fl_Text_Buffer *)
```

Pointer to a function called after reading a non UTF-8 encoded file.

This function is called after reading a file if the file content was transcoded to UTF-8. Its default implementation calls fl_alert() with the text of file_encoding_warning_message. No warning message is displayed if this pointer is set to NULL. Use input_file_was_transcoded to be informed if file input required transcoding to UTF-8.

The documentation for this class was generated from the following files:

- Fl_Text_Buffer.H
- Fl_Text_Buffer.cxx

### 12.143 Fl_Text_Display Class Reference

Rich text display widget.

```c
#include <Fl_Text_Display.H>
```

Inheritance diagram for Fl_Text_Display:
Classes

- struct Style_Table_Entry

  This structure associates the color, font, and font size of a string to draw with an attribute mask matching attr.

Public Types

- enum {
  NORMAL_CURSOR, CARET_CURSOR, DIM_CURSOR, BLOCK_CURSOR, HEAVY_CURSOR, SIMPLE_CURSOR
}

  text display cursor shapes enumeration

- enum { CURSOR_POS, CHARACTER_POS }

  the character position is the left edge of a character, whereas the cursor is thought to be between the centers of two consecutive characters.

- enum {
  DRAG_NONE = -2, DRAG_START_DND = -1, DRAG_CHAR = 0, DRAG_WORD = 1, DRAG_LINE = 2
}

  drag types - they match Fl::event_clicks() so that single clicking to start a collection selects by character, double clicking selects by word and triple clicking selects by line.

- enum { WRAP_NONE, WRAP_AT_COLUMN, WRAP_AT_PIXEL, WRAP_AT_BOUNDS }

  wrap types - used in wrap_mode()

- enum {
  ATTR_BGCOLOR = 0x0001, ATTR_BGCOLOR_EXT = 0x0002, ATTR_BGCOLOR_EXT_ = 0x0003, ATTR_UNDERLINE = 0x0004,
  ATTR_GRAMMAR = 0x0008, ATTR_SPELLING = 0x000C, ATTR_STRIKE_THROUGH = 0x0010, ATTR_LINES_MASK = 0x001C
}

  attribute flags in Style_Table_Entry.attr

- typedef void(* Unfinished_Style_Cb) (int, void *)

Public Member Functions

- Fl_Text_Buffer * buffer () const

  Gets the current text buffer associated with the text widget.

- void buffer (Fl_Text_Buffer &buf)

  Sets the current text buffer associated with the text widget.

- void buffer (Fl_Text_Buffer *buf)

  Attach a text buffer to display, replacing the current buffer (if any).

- double col_to_x (double col) const

  Convert a column number into an x pixel position.

- int count_lines (int start, int end, bool start_pos_is_line_start) const

  Count the number of lines between two positions.

- Fl_Color cursor_color () const

  Gets the text cursor color.

- void cursor_color (Fl_Color n)
Sets the text cursor color.

- int **cursor_style** () const
- void **cursor_style** (int style)
  
  Sets the text cursor style.

- Fl_Text_Display (int X, int Y, int W, int H, const char *l=0)
  
  Creates a new text display widget.

- Fl_Color **grammar_underline_color** () const
  
  Gets the underline color for style attribute ATTR_GRAMMAR.

- void **grammar_underline_color** (Fl_Color color)
  
  Sets the underline color for style attribute ATTR_GRAMMAR.

- int **handle** (int e) FL_OVERRIDE
  
  Event handling.

- void **hide_cursor** ()
  
  Hides the text cursor.

- void **highlight_data** (Fl_Text_Buffer *styleBuffer, const Style_Table_Entry *styleTable, int nStyles, char unfinishedStyle, Unfinished_Style_Cb unfinishedHighlightCB, void *cbArg)
  
  Attach (or remove) highlight information in text display and redisplay.

- int **in_selection** (int x, int y) const
  
  Check if a pixel position is within the primary selection.

- void **insert** (const char *text)
  
  Inserts "text" at the current cursor location.

- int **insert_position** () const
  
  Gets the position of the text insertion cursor for text display.

- void **insert_position** (int newPos)
  
  Sets the position of the text insertion cursor for text display.

- int **line_end** (int startPos, bool startPosIsLineStart) const
  
  Returns the end of a line.

- int **line_start** (int pos) const
  
  Return the beginning of a line.

- Fl_Align **linenumber_align** () const
  
  Returns the alignment used for line numbers (if enabled).

- void **linenumber_align** (Fl_Align val)
  
  Set alignment for line numbers (if enabled).

- Fl_Color **linenumberbgcolor** () const
  
  Returns the background color used for line numbers (if enabled).

- void **linenumberbgcolor** (Fl_Color val)
  
  Set the background color used for line numbers (if enabled).

- Fl_Color **linenumberfgcolor** () const
  
  Return the foreground color used for line numbers (if enabled).

- void **linenumberfgcolor** (Fl_Color val)
  
  Set the foreground color used for line numbers (if enabled).

- Fl_Font **linenumberfont** () const
  
  Return the font used for line numbers (if enabled).

- void **linenumberfont** (Fl_Font val)
  
  Set the font used for line numbers (if enabled).

- const char * **linenumberformat** () const
  
  Returns the line number printf() format string.

- void **linenumberformat** (const char *val)
  
  Sets the printf() style format string used for line numbers.

- Fl_Fontsize **linenumbersize** () const
  
  Return the font size used for line numbers (if enabled).
• void linenumber_size (Fl_Fontsize val)
  Set the font size used for line numbers (if enabled).

• int linenumber_width () const
  Return the screen area width provided for line numbers.

• void linenumber_width (int width)
  Set width of screen area for line numbers.

• int move_down ()
  Moves the current insert position down one line.

• int move_left ()
  Moves the current insert position left one character.

• int move_right ()
  Moves the current insert position right one character.

• int move_up ()
  Moves the current insert position up one line.

• void next_word (void)
  Moves the current insert position right one word.

• void overstrike (const char ∗text)
  Replaces text at the current insert position.

• int position_style (int lineStartPos, int lineLen, int lineIndex) const
  Find the correct style for a character.

• int position_to_xy (int pos, int ∗x, int ∗y) const
  Convert a character index into a pixel position.

• void previous_word (void)
  Moves the current insert position left one word.

• virtual void recalc_display ()
  Recalculate the display's visible lines and scrollbar sizes.

• void redisplay_range (int start, int end)
  Marks text from start to end as needing a redraw.

• void resize (int X, int Y, int W, int H) FL_OVERRIDE
  Change the size of the displayed text area.

• int rewind_lines (int startPos, int nLines)
  Skip a number of lines back.

• void scroll (int topLineNum, int horizOffset)
  Scrolls the current buffer to start at the specified line and column.

• Fl_Align scrollbar_align () const
  Gets the scrollbar alignment type.

• void scrollbar_align (Fl_Align a)
  Sets the scrollbar alignment type.

• int scrollbar_size () const
  Gets the current size of the scrollbars' troughs, in pixels.

• void scrollbar_size (int newSize)
  Sets the pixel size of the scrollbars' troughs to newSize, in pixels.

• int scrollbar_width () const
  Returns the global value Fl::scrollbar_size() unless a specific scrollbar_width_ has been set.

• void scrollbar_width (int width)
  Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

• Fl_Color secondary_selection_color () const
  Gets the background color for the secondary selection block.

• void secondary_selection_color (Fl_Color color)
  Sets the background color for the secondary selection block.

• int shortcut () const
• void shortcut (int s)
• void show_cursor (int b=1)
  Shows the text cursor.
• void show_insert_position ()
  Scrolls the text buffer to show the current insert position.
• int skip_lines (int startPos, int nLines, bool startPosIsLineStart)
  Skip a number of lines forward.
• Fl_Color spelling_underline_color () const
  Gets the underline color for style attribute ATTR_SPELLING.
• void spelling_underline_color (Fl_Color color)
  Sets the underline color for style attribute ATTR_SPELLING.
• Fl_Text_Buffer * style_buffer () const
  Gets the current style buffer associated with the text widget.
• Fl_Color textcolor () const
  Gets the default color of text in the widget.
• void textcolor (Fl_Color n)
  Sets the default color of text in the widget.
• Fl_Font textfont () const
  Gets the default font used when drawing text in the widget.
• void textfont (Fl_Font s)
  Sets the default font used when drawing text in the widget.
• Fl_Fontsize textsize () const
  Gets the default size of text in the widget.
• void textsize (Fl_Fontsize s)
  Sets the default size of text in the widget.
• int word_end (int pos) const
  Moves the insert position to the end of the current word.
• int word_start (int pos) const
  Moves the insert position to the beginning of the current word.
• void wrap_mode (int wrap, int wrap_margin)
  Set the new text wrap mode.
• int wrapped_column (int row, int column) const
  Nobody knows what this function does.
• int wrapped_row (int row) const
  Nobody knows what this function does.
• double x_to_col (double x) const
  Convert an x pixel position into a column number.
• ~Fl_Text_Display ()
  Free a text display and release its associated memory.

Public Member Functions inherited from Fl_Group

• Fl_Widget * & _ddfdesign_kludge ()
  This is for forms compatibility only.
• void add (Fl_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.
• void add (Fl_Widget &w)
  See void Fl_Group::add(Fl_Widget &w)
• void add_resizable (Fl_Widget &o)
  Adds a widget to the group and makes it the resizable widget.
• Fl_Widget * const * array () const
Returns a pointer to the array of children.

- Fl_Group const * as_group () const FL_OVERRIDE
- Fl_Group const * as_group () FL_OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()
  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const
  Returns array()[n].
- int children () const
  Returns how many child widgets the group has.
- void clear ()
  Deletes all child widgets from memory recursively.
- unsigned int clip_children ()
  Returns the current clipping mode.
- void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.
- void end ()
  Exactly the same as current(this->parent()).
- int find (const Fl_Widget &o) const
  See int Fl_Group::find(const Fl_Widget *w) const.
- int find (const Fl_Widget *) const
  Searches the child array for the widget and returns the index.
- Fl_Group (int, int, int, int, const char * =0)
  Creates a new Fl_Group widget using the given position, size, and label string.
- void focus (Fl_Widget *W)
- void forms_end ()
  This is for forms compatibility only.
- void init_sizes ()
  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)
  This does insert(w, find(before)).
- void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)
  Removes the widget o from the group.
- void remove (int index)
  Removes the widget at index from the group but does not delete it.
- Fl_Widget * resizable () const
  Returns the group’s resizable widget.
- void resizable (Fl_Widget &o)
  Sets the group’s resizable widget.
- void resizable (Fl_Widget *o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()
  The destructor also deletes all the children.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ as_gl_window () const
- virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ as_window () const
- void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
  Checks if the widget value changed since the last callback.
- void clear_active ()
Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  Clears or sets the damage flags.

- void clear_output ()
  Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.

- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

- Fl_Color color () const
  Gets the background color of the widget.

- void color (Fl_Color bg)
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

- Fl_Color color2 () const
  For back compatibility only.

- void color2 (unsigned a)
  For back compatibility only.

- int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.

- void copy_label (const char *new_label)
  Sets the current label.

- void copy_tooltip (const char *text)
  Sets the current tooltip text.

- uchar damage () const
  Returns non-zero if draw() needs to be called.

- void damage (uchar c)
  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- int damage_resize (int, int, int, int)
  Internal use only.

- void deactivate ()
  Deactivates the widget.

- Fl_Image * deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

- const Fl_Image * deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

- void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

- void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with arbitrary arguments.

• void draw_label (int, int, int, int, Fl_Align) const
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• int h () const
  
  Gets the widget height.

• virtual void hide ()
  
  Makes a widget invisible.

• Fl_Image * image ()
  
  Gets the image that is used as part of the widget label when in the active state.

• const Fl_Image * image () const
  
  Gets the image that is used as part of the widget label when in the active state.

• void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the active state.

• void image (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the active state.

• int image_bound () const
  
  Returns whether the image is managed by the widget.

• int inside (const Fl_Widget *wgt) const
  
  Checks if this widget is a child of wgt.

• int is_label_copied () const
  
  Returns whether the current label was assigned with copy_label().

• const char * label () const
  
  Gets the current label text.

• void label (const char *text)
  
  Sets the current label pointer.

• void label (Fl_Labeltype a, const char *b)
  
  Shortcut to set the label text and type in one call.

• Fl_Color labelcolor () const
  
  Gets the label color.

• void labelcolor (Fl_Color c)
  
  Sets the label color.

• Fl_Font labelfont () const
  
  Gets the font to use.

• void labelfont (Fl_Font f)
  
  Sets the font to use.

• Fl_Fontsize labelsize () const
  
  Gets the font size in pixels.

• void labelsize (Fl_Fontsize pix)
  
  Sets the font size in pixels.

• Fl_Labeltype labeltype () const
  
  Gets the label type.

• void labeltype (Fl_Labeltype a)
  
  Sets the label type.

• void measure_label (int &ww, int &hh) const
  
  Sets width ww and height hh accordingly with the label size.

• bool needs_keyboard () const
  
  Returns whether this widget needs a keyboard.

• void needs_keyboard (bool needs)
  
  Sets whether this widget needs a keyboard.

• unsigned int output () const
Returns if a widget is used for output only.

- **Fl_Group** ∗ parent () const
  
  Returns a pointer to the parent widget.

- void parent (Fl_Group ∗p)
  
  Internal use only - "for hacks only".

- void position (int X, int Y)
  
  Repositions the window or widget.

- void redraw ()
  
  Schedules the drawing of the widget.

- void redraw_label ()
  
  Schedules the drawing of the label.

- Fl_Color selection_color () const
  
  Gets the selection color.

- void selection_color (Fl_Color a)
  
  Sets the selection color.

- void set_active ()
  
  Marks the widget as active without sending events or changing focus.

- void set_changed ()
  
  Marks the value of the widget as changed.

- void set_output ()
  
  Sets a widget to output only.

- void set_visible ()
  
  Makes the widget visible.

- void set_visible_focus ()
  
  Enables keyboard focus navigation with this widget.

- int shortcut_label () const
  
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.

- void shortcut_label (int value)
  
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.

- virtual void show ()
  
  Makes a widget visible.

- void size (int W, int H)
  
  Changes the size of the widget.

- int take_focus ()
  
  Gives the widget the keyboard focus.

- unsigned int takesevents () const
  
  Returns if the widget is able to take events.

- int testShortcut ()
  
  Returns true if the widget’s label contains the entered ‘&x’ shortcut.

- const char ∗ tooltip () const
  
  Gets the current tooltip text.

- void tooltip (const char ∗text)
  
  Sets the current tooltip text.

- Fl_Window ∗ top_window () const
  
  Returns a pointer to the top-level window for the widget.

- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const
  
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const
  
  Gets the widget type.

- void type (uchar t)
  
  Sets the widget type.
• **int** `use_accents_menu()`  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• **void** `user_data()` const  
  Gets the user data for this widget.

• **void** `user_data(Fl_Callback_User_Data *v, bool auto_free)`  
  Sets the user data for this widget.

• **void** `user_data(void *v)`  
  Sets the user data for this widget.

• **unsigned int** `visible()` const  
  Returns whether a widget is visible.

• **unsigned int** `visible_focus()` const  
  Checks whether this widget has a visible focus.

• **void** `visible_focus(int v)`  
  Modifies keyboard focus navigation.

• **int** `visible_r()` const  
  Returns whether a widget and all its parents are visible.

• **int** `w()` const  
  Gets the widget width.

• **Fl_When** `when()` const  
  Returns the conditions under which the callback is called.

• **void** `when(uchar i)`  
  Sets the flags used to decide when a callback is called.

• **Fl_Window ** `window()` const  
  Returns a pointer to the nearest parent window up the widget hierarchy.

• **int** `x()` const  
  Gets the widget position in its window.

• **int** `y()` const  
  Gets the widget position in its window.

• virtual **`~Fl_Widget()`**  
  Destroys the widget.

**Protected Types**

• **enum** {
  `DRAW_LINE`, `FIND_INDEX`, `FIND_INDEX_FROM_ZERO`, `GET_WIDTH`, `FIND_CURSOR_INDEX`}

**Protected Types inherited from Fl_Widget**

• **enum** {
  `INACTIVE = 1<<0`, `INVISIBLE = 1<<1`, `OUTPUT = 1<<2`, `NOBORDER = 1<<3`,
  `FORCE_POSITION = 1<<4`, `NON_MODAL = 1<<5`, `SHORTCUT_LABEL = 1<<6`, `CHANGED = 1<<7`,
  `OVERRIDE = 1<<8`, `VISIBLE_FOCUS = 1<<9`, `COPYED_LABEL = 1<<10`, `CLIP_CHILDREN = 1<<11`,
  `MENU_WINDOW = 1<<12`, `TOOLTIP_WINDOW = 1<<13`, `MODAL = 1<<14`, `NO_OVERLAY = 1<<15`,
  `GROUP_RELATIVE = 1<<16`, `COPYED_TOOLTIP = 1<<17`, `FULLSCREEN = 1<<18`,
  `MAC_USE_ACCENTS_MENU = 1<<19`,
  `NEEDS_KEYBOARD = 1<<20`, `IMAGE_BOUND = 1<<21`, `DEIMAGE_BOUND = 1<<22`,
  `AUTO_DELETE_USER_DATA = 1<<23`,
  `MAXIMIZED = 1<<24`, `USERFLAG3 = 1<<29`, `USERFLAG2 = 1<<30`, `USERFLAG1 = 1<<31`}

  flags possible values enumeration.
Protected Member Functions

- `void absolute_top_line_number(int oldFirstChar)`
  
  Re-calculate absolute top line number for a change in scroll position.

- `void calc_last_char()`
  
  Update last display character index.

- `void calc_line_starts(int startLine, int endLine)`
  
  Update the line starts array.

- `void clear_rect(int style, int x, int y, int width, int height) const`
  
  Clear a rectangle with the appropriate background color for `style`.

- `void display_insert()`
  
  Scroll the display to bring insertion cursor into view.

- `void draw() FL_OVERRIDE`
  
  Draw the widget.

- `void draw_cursor(int, int)`
  
  Draw a cursor with top center at `X`, `Y`.

- `void draw_line_numbers(bool clearAll)`
  
  Refresh the line number area.

- `void draw_range(int start, int end)`
  
  Draw a range of text.

- `void draw_string(int style, int x, int y, int toX, const char *string, int nChars) const`
  
  Draw a text segment in a single style.

- `void draw_text(int X, int Y, int W, int H)`
  
  Refresh a rectangle of the text display.

- `void draw_vline(int visLineNum, int leftClip, int rightClip, int leftCharIndex, int rightCharIndex)`
  
  Draw a single line of text.

- `int empty_vlines() const`
  
  Return true if there are lines visible with no corresponding buffer text.

- `void extend_range_for_styles(int *start, int *end)`
  
  I don't know what this does!

- `void find_line_end(int pos, bool start_pos_is_line_start, int *lineEnd, int *nextLineStart) const`
  
  Finds both the end of the current line and the start of the next line.

- `void find_wrap_range(const char *deletedText, int pos, int nInserted, int nDeleted, int *modRangeStart, int *modRangeEnd, int *linesInserted, int *linesDeleted)`
  
  Wrapping calculations.

- `int find_x(const char *s, int len, int style, int x) const`
  
  Find the index of the character that lies at the given x position / closest cursor position.

- `int get_absolute_top_line_number() const`
  
  Returns the absolute (non-wrapped) line number of the first line displayed.

- `int handle_rmb(int readonly)`
  
  Handle right mouse button down events.

- `int handle_vline(int mode, int lineStart, int lineLen, int leftChar, int rightChar, int topClip, int bottomClip, int leftClip, int rightClip) const`
  
  Universal pixel machine.

- `int longest_vline() const`
  
  Find the longest line of all visible lines.

- `void maintain_absolute_top_line_number(int state)`
  
  Line numbering stuff, currently unused.

- `int maintaining_absolute_top_line_number() const`
  
  Returns true if a separate absolute top line number is being maintained.

- `void measure_deleted_lines(int pos, int nDeleted)`
  
  Generated by Doxygen
Wrapping calculations.

- **double measure_proportional_character** (const char *s, int colNum, int pos) const
  Wrapping calculations.

- **int measure_vline** (int visLineNum) const
  Returns the width in pixels of the displayed line pointed to by "visLineNum".

- **void offset_line_starts** (int newTopLineNum)
  Offset line start counters for a new vertical scroll position.

- **int position_to_line** (int pos, int *lineNum) const
  Convert a position index into a line number offset.

- **int position_to_linecol** (int pos, int *lineNum, int *column) const
  Find the line and column number of position pos.

- **void reset_absolute_top_line_number** ()
  Reestablish the absolute (non-wrapped) top line number.

- **int scroll_** (int topLineNum, int horizOffset)
  Scrolls the current buffer to start at the specified line and column.

- **double string_width** (const char *string, int length, int style) const
  Find the width of a string in the font of a particular style.

- **void update_h_scrollbar** ()
  Update horizontal scrollbar.

- **void update_line_starts** (int pos, int charsInserted, int charsDeleted, int linesInserted, int linesDeleted, int scrolled)
  Update line start arrays and variables.

- **void update_v_scrollbar** ()
  Update vertical scrollbar.

- **int vline_length** (int visLineNum) const
  Count number of bytes in a visible line.

- **int wrap_uses_character** (int lineEndPos) const
  Check if the line break is caused by a newline or by line wrapping.

- **void wrapped_line_counter** (Fl_Text_Buffer *buf, int startPos, int maxPos, int maxLines, bool startPosIsLineStart, int styleBufOffset, int *retPos, int *retLines, int *retLineStart, int *retLineEnd, bool countLastLineMissingNewLine=true) const
  Wrapping calculations.

- **int xy_to_position** (int x, int y, int PosType=CHARACTER_POS) const
  Translate a pixel position into a character index.

- **void xy_to_rowcol** (int x, int y, int *row, int *column, int PosType=CHARACTER_POS) const
  Translate pixel coordinates into row and column.

Protected Member Functions inherited from Fl_Group

- **Fl_Rect * bounds** ()
  Returns the internal array of widget sizes and positions.

- **void draw_child** (Fl_Widget &widget) const
  Forces a child to redraw.

- **void draw_children** ()
  Draws all children of the group.

- **void draw_outside_label** (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.

- **virtual int on_insert** (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.

- **virtual int on_move** (int, int)
  Allow derived groups to act when a widget is moved within the group.

- **virtual void on_remove** (int)
Allow derived groups to act when a child widget is removed from the group.

- **int * sizes ()**
  Returns the internal array of widget sizes and positions.

- **void update_child (FL_Widget &widget) const**
  Draws a child only if it needs it.

**Protected Member Functions inherited from FL_Widget**

- **void clear_flag (unsigned int c)**
  Clears a flag in the flags mask.

- **void draw_backdrop () const**
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- **void draw_box () const**
  Draws the widget box according its box style.

- **void draw_box (FL_Boxtype t, FL_Color c) const**
  Draws a box of type t, of color c at the widget's position and size.

- **void draw_box (FL_Boxtype t, int x, int y, int w, int h, FL_Color c) const**
  Draws a box of type t, of color c at the position X,Y and size W,H.

- **void draw_focus () const**
  Draws a focus rectangle around the widget.

- **void draw_focus (FL_Boxtype t, int X, int Y, int W, int H) const**
  Draws a focus rectangle around the widget.

- **void draw_focus (FL_Boxtype t, int x, int y, int w, int h, FL_Color bg) const**
  Draws a focus box for the widget at the given position and size.

- **void draw_label () const**
  Draws the widget's label at the defined label position.

- **void draw_label (int, int, int, int) const**
  Draws the label in an arbitrary bounding box.

- **FL_Widget (int x, int y, int w, int h, const char *label=0L)**
  Creates a widget at the given position and size.

- **unsigned int flags () const**
  Gets the widget flags mask.

- **void h (int v)**
  Internal use only.

- **void set_flag (unsigned int c)**
  Sets a flag in the flags mask.

- **void w (int v)**
  Internal use only.

- **void x (int v)**
  Internal use only.

- **void y (int v)**
  Internal use only.

**Static Protected Member Functions**

- **static void buffer_modified_cb (int pos, int nInserted, int nDeleted, int nRestyled, const char *deletedText, void *cbArg)**
  This is called whenever the buffer is modified.

- **static void buffer_predelete_cb (int pos, int nDeleted, void *cbArg)**
  This is called before any characters are deleted.

- **static void h_scrollbar_cb (FL_Scrollbar *w, FL_Text_Display *d)**
  Callback for drag or valueChanged on horizontal scrollbar.
• static void `scroll_timer_cb` (void *)
  
  Timer callback for scroll events.

• static void `v_scrollbar_cb` (Fl_Scrollbar *w, Fl_Text_Display *d)
  
  Callback for drag or valueChanged on vertical scrollbar.

Protected Attributes

• int `damage_range1_end`
• int `damage_range1_start`
• int `damage_range2_end`
• int `damage_range2_start`
• int `display_insert_position_hint`
• int `dragging`
• int `dragPos`
• int `dragType`
• Fl_Color `grammar_underline_color_`
• Fl_Align `linenumber_align_`
• Fl_Color `linenumberbgcolor_`
• Fl_Color `linenumberbgcolor_`
• Fl_Font `linenumber_font_`
• const char * `linenumber_format_`
• Fl_Fontsize `linenumber_size_`
• int `mAbsTopLineNum`
• Fl_Text_Buffer * `mBuffer`
• double `mColumnScale`
• int `mContinuousWrap`
• Fl_Color `mCursor_color`
• int `mCursorOldY`
• int `mCursorOn`
• int `mCursorPos`
• int `mCursorPreferredXPos`
• int `mCursorStyle`
• int `mCursorToHint`
• int `mFirstChar`
• void * `mHighlightCBArg`
• int `mHorizOffset`
• int `mHorizOffsetHint`
• Fl_Scrollbar * `mHScrollbar`
• int `mLastChar`
• int `mLineNumLeft`
• int `mLineNumWidth`
• int * `mLineStarts`
• int `mMaxsize`
• int `mModifyingTabDistance`
• int `mNBufferLines`
• int `mNeedAbsTopLineNum`
• int `mNLinesDeleted`
• int `mNStyles`
• int `mNVisibleLines`
• Fl_Text_Buffer * `mStyleBuffer`
• const Style_Table_Entry * `mStyleTable`
• int `mSuppressResync`
• int `mTopLineNum`
• int `mTopLineNumHint`
• Unfinished_Style_Cb `mUnfinishedHighlightCB`
Rich text display widget.

This is the FLTK text display widget. It allows the user to view multiple lines of text and supports highlighting, word wrap, mixes of font faces and colors, line numbers and scrolling. The buffer that is displayed in the widget is managed by the Fl_Text_Buffer class. A single Text Buffer can be displayed by multiple Text Displays.
**Example Use**

```cpp
#include <FL/Fl_Text_Display.H>

int main() {
  Fl_Text_Buffer *buff = new Fl_Text_Buffer();
  Fl_Text_Display *disp = new Fl_Text_Display(10, 10, 640, 480);
  disp->buffer(buff); // attach text buffer to display widget
  buff->text("line one
  \nline two"); // add some text to buffer
}
```

**Features**

- Word wrap: `wrap_mode()`, `wrapped_column()`, `wrapped_row()`
- Font control: `textfont()`, `textsize()`, `textcolor()`
• Font styling: `highlight_data()`

• Cursor: `cursor_style()`, `show_cursor()`, `hide_cursor()`, `cursor_color()`

• Line numbers: `linenumber_width()`, `linenumber_font()`, `linenumber_size()`, `linenumberbgcolor()`, `linenumberbgcolor()`,
  `linenumber_align()`, `linenumberformat()`

Note that other features may be available via `Fl_Text_Editor` and `Fl_Text_Buffer` classes.

Note

Line numbers were added in FLTK 1.3.3.

See also

`Fl_Widget::shortcut_label(int)`

12.143.2 Member Enumeration Documentation

12.143.2.1 anonymous enum

anonymous enum
text display cursor shapes enumeration

Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL_CURSOR</td>
<td>I-beam.</td>
</tr>
<tr>
<td>CARET_CURSOR</td>
<td>caret under the text</td>
</tr>
<tr>
<td>DIM_CURSOR</td>
<td>dim I-beam</td>
</tr>
<tr>
<td>BLOCK_CURSOR</td>
<td>unfille box under the current character</td>
</tr>
<tr>
<td>HEAVY_CURSOR</td>
<td>thick I-beam</td>
</tr>
<tr>
<td>SIMPLE_CURSOR</td>
<td>as cursor as <code>Fl_Input</code> cursor</td>
</tr>
</tbody>
</table>

12.143.2.2 anonymous enum

anonymous enum
wrap types - used in `wrap_mode()`

Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRAP_NONE</td>
<td>don't wrap text at all</td>
</tr>
<tr>
<td>WRAP_AT_COLUMN</td>
<td>wrap text at the given text column</td>
</tr>
<tr>
<td>WRAP_AT_PIXEL</td>
<td>wrap text at a pixel position</td>
</tr>
<tr>
<td>WRAP_AT_BOUNDS</td>
<td>wrap text so that it fits into the widget width</td>
</tr>
</tbody>
</table>

12.143.2.3 anonymous enum

anonymous enum
attribute flags in `Style_Table_Entry.attr`

Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_BGCOLOR</td>
<td>use the background color in the bgcolor field</td>
</tr>
<tr>
<td>ATTR_BGCOLOR_EXT</td>
<td>(internal use)</td>
</tr>
<tr>
<td>ATTR_BGCOLOR_EXT</td>
<td>extend background color to the end of the line</td>
</tr>
<tr>
<td>ATTR_UNDERLINE</td>
<td>a single underline, underline types are mutually exclusive</td>
</tr>
</tbody>
</table>
12.143.3 Constructor & Destructor Documentation

12.143.3.1 Fl_Text_Display()

```cpp
Fl_Text_Display::Fl_Text_Display ( 
    int X, 
    int Y, 
    int W, 
    int H, 
    const char ∗ l = 0 )
```

Creates a new text display widget.

**Parameters**

<table>
<thead>
<tr>
<th>X,Y,W,H</th>
<th>position and size of widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>label text, defaults to none</td>
</tr>
</tbody>
</table>

12.143.3.2 ~Fl_Text_Display()

```cpp
Fl_Text_Display::~Fl_Text_Display ( )
```

Free a text display and release its associated memory.

**Note**

The text buffer that the text display displays is a separate entity and is not freed, nor are the style buffer or style table.

**See also**

`Fl_Text_Display::buffer(Fl_Text_Buffer ∗ buf)`

12.143.4 Member Function Documentation

12.143.4.1 absolute_top_line_number()

```cpp
void Fl_Text_Display::absolute_top_line_number ( 
    int oldFirstChar ) [protected]
```

Re-calculate absolute top line number for a change in scroll position. Does nothing if the absolute top line number is not being maintained.

12.143.4.2 buffer() [1/3]

```cpp
Fl_Text_Buffer ∗ Fl_Text_Display::buffer ( ) const [inline]
```

Gets the current text buffer associated with the text widget. Multiple text widgets can be associated with the same text buffer.

**Returns**

- current text buffer
See also

```cpp
Fl_Text_Display::buffer(Fl_Text_Buffer* buf)
Fl_Text_Display::buffer(Fl_Text_Buffer& buf)
```

### 12.143.4.3 buffer() [2/3]

```cpp
void Fl_Text_Display::buffer (Fl_Text_Buffer & buf) [inline]
```

Sets the current text buffer associated with the text widget.
Multiple text widgets can be associated with the same text buffer.

**Parameters**

- `buf` new text buffer

See also

```cpp
Fl_Text_Display::buffer(Fl_Text_Buffer* buf)
```

### 12.143.4.4 buffer() [3/3]

```cpp
void Fl_Text_Display::buffer (Fl_Text_Buffer * buf)
```

Attach a text buffer to display, replacing the current buffer (if any).
Multiple text widgets can be associated with the same text buffer.

**Note**

The caller is responsible for the old (replaced) buffer (if any). This method does not delete the old buffer.

**Parameters**

- `buf` attach this text buffer

### 12.143.4.5 buffer_modified_cb()

```cpp
void Fl_Text_Display::buffer_modified_cb (int pos, int nInserted, int nDeleted, int nRestyled, const char * deletedText, void * cbArg) [static], [protected]
```

This is called whenever the buffer is modified.
Callback attached to the text buffer to receive modification information.
This callback can be used to adjust the display or update other setting. It is not advisable to change any buffers or
text in this callback, or line counting may get out of sync.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pos</code></td>
<td>starting index of modification</td>
</tr>
<tr>
<td><code>nInserted</code></td>
<td>number of bytes we inserted (must be UTF-8 aligned!)</td>
</tr>
<tr>
<td><code>nDeleted</code></td>
<td>number of bytes deleted (must be UTF-8 aligned!)</td>
</tr>
<tr>
<td><code>nRestyled</code></td>
<td>??</td>
</tr>
<tr>
<td><code>deletedText</code></td>
<td>this is what was removed, must not be NULL if nDeleted is set</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>cbArg</em></td>
<td>&quot;this&quot; pointer for static callback function</td>
</tr>
</tbody>
</table>

#### 12.143.4.6 buffer_predelete_cb()

```c
void Fl_Text_Display::buffer_predelete_cb ( int pos, int nDeleted, void * cbArg ) [static], [protected]
```

This is called before any characters are deleted. Callback attached to the text buffer to receive delete information before the modifications are actually made. This callback can be used to adjust the display or update other setting. It is not advisable to change any buffers or text in this callback, or line counting may get out of sync.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>pos</em></td>
<td>starting index of deletion</td>
</tr>
<tr>
<td><em>nDeleted</em></td>
<td>number of bytes we will delete (must be UTF-8 aligned!)</td>
</tr>
<tr>
<td><em>cbArg</em></td>
<td>&quot;this&quot; pointer for static callback function</td>
</tr>
</tbody>
</table>

#### 12.143.4.7 calc_last_char()

```c
void Fl_Text_Display::calc_last_char ( ) [protected]
```

Update last display character index. Given a Fl_Text_Display with a complete, up-to-date lineStarts array, update the lastChar entry to point to the last buffer position displayed.

#### 12.143.4.8 calc_line_starts()

```c
void Fl_Text_Display::calc_line_starts ( int startLine, int endLine ) [protected]
```

Update the line starts array. Scan through the text in the Text Display's buffer and recalculate the line starts array values beginning at index "startLine" and continuing through (including) "endLine". It assumes that the line starts entry preceding "startLine" (or mFirstChar if startLine is 0) is good, and re-counts newlines to fill in the requested entries. Out of range values for "startLine" and "endLine" are acceptable.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>startLine</em>, <em>endLine</em></td>
<td>range of lines to scan as line numbers</td>
</tr>
</tbody>
</table>

#### 12.143.4.9 clear_rect()

```c
void Fl_Text_Display::clear_rect ( int style, int X, int Y, int width, int height ) const [protected]
```

Clear a rectangle with the appropriate background color for *style*. 
### Parameters

<table>
<thead>
<tr>
<th>style</th>
<th>index into style table</th>
</tr>
</thead>
<tbody>
<tr>
<td>X,Y, width, height</td>
<td>size and position of background area</td>
</tr>
</tbody>
</table>

#### 12.143.4.10 col_to_x()  

```cpp
double Fl_Text_Display::col_to_x (double col) const
```

Convert a column number into an x pixel position.

**Parameters**
- `col` an approximate column number based on the main font

**Returns**
- number of pixels from the left margin to the left of an average sized character

**See also**
- `x_to_col()`

#### 12.143.4.11 count_lines()  

```cpp
int Fl_Text_Display::count_lines (int startPos, int endPos, bool startPosIsLineStart) const
```

Count the number of lines between two positions.

Same as `Fl_Text_Buffer::count_lines()`, but takes into account wrapping if wrapping is turned on. If the caller knows that `startPos` is at a line start, it can pass `startPosIsLineStart` as True to make the call more efficient by avoiding the additional step of scanning back to the last newline.

**Parameters**
- `startPos` index to first character
- `endPos` index after last character
- `startPosIsLineStart` avoid scanning back to the line start

**Returns**
- number of lines

#### 12.143.4.12 cursor_color() [1/2]

```cpp
Fl_Color Fl_Text_Display::cursor_color ( ) const [inline]
```

Gets the text cursor color.

**Returns**
- cursor color

#### 12.143.4.13 cursor_color() [2/2]

```cpp
void Fl_Text_Display::cursor_color ( Fl_Color n ) [inline]
```

Generated by Doxygen
Sets the text cursor color.

Parameters

\[ n \] new cursor color

12.143.4.14 cursor_style()

void Fl_Text_Display::cursor_style {
    int style
}

Sets the text cursor style.
Sets the text cursor style to one of the following:

- \texttt{Fl_Text_Display::NORMAL_CURSOR} - Shows an I beam.
- \texttt{Fl_Text_Display::CARET_CURSOR} - Shows a caret under the text.
- \texttt{Fl_Text_Display::DIM_CURSOR} - Shows a dimmed I beam.
- \texttt{Fl_Text_Display::BLOCK_CURSOR} - Shows an unfilled box around the current character.
- \texttt{Fl_Text_Display::HEAVY_CURSOR} - Shows a thick I beam.

This call also switches the cursor on and may trigger a redraw.

Parameters

\[ \text{style} \] new cursor style

12.143.4.15 display_insert()

void Fl_Text_Display::display_insert ( ) [protected]

Scroll the display to bring insertion cursor into view.
Note: it would be nice to be able to do this without counting lines twice (scroll_() counts them too) and/or to count
from the most efficient starting point, but the efficiency of this routine is not as important to the overall performance
of the text display.

12.143.4.16 draw()

void Fl_Text_Display::draw (void ) [protected], [virtual]

Draw the widget.
This function tries to limit drawing to smaller areas if possible.
Reimplemented from \texttt{Fl_Group}.

12.143.4.17 draw_cursor()

void Fl_Text_Display::draw_cursor {
    int X,
    int Y ) [protected]

Draw a cursor with top center at \( X, Y \).

Parameters

\[ X,Y \] cursor position in pixels
12.143.4.18  **draw_line_numbers()**

```cpp
void Fl_Text_Display::draw_line_numbers ( bool clearAll ) [protected]
```

Refresh the line number area.

**Parameters**

| clearAll | (currently unused) If False, only draws the line number text, does not clear the area behind it. If True, clears the area and redraws the text. Use False to avoid a 'flash' for single buffered windows. |

12.143.4.19  **draw_range()**

```cpp
void Fl_Text_Display::draw_range ( int startPos, int endpos ) [protected]
```

Draw a range of text.

Refresh all of the text between buffer positions `startpos` and `endpos` not including the character at the position `endpos`.

If `endpos` points beyond the end of the buffer, refresh the whole display after `startpos`, including blank lines which are not technically part of any range of characters.

**Parameters**

| startPos | index of first character to draw |
| endpos   | index after last character to draw |

12.143.4.20  **draw_string()**

```cpp
void Fl_Text_Display::draw_string ( int style, int X, int Y, int toX, const char * string, int nChars ) const [protected]
```

Draw a text segment in a single style.

Draw a string or blank area according to parameter `style`, using the appropriate colors and drawing method for that style, with top left corner at `X`, `Y`. If style says to draw text, use `string` as source of characters, and draw `nChars`. If style is FILL, erase rectangle where text would have drawn from `X` to `toX` and from `Y` to the maximum y extent of the current font(s).

**Parameters**

| style    | index into style lookup table |
| X,Y      | drawing origin                |
| toX      | rightmost position if this is a fill operation |
| string   | text if this is a drawing operation |
| nChars   | number of characters to draw  |

12.143.4.21  **draw_text()**

```cpp
void Fl_Text_Display::draw_text ( int left, 
```

Generated by Doxygen
Refresh a rectangle of the text display.

Parameters

| left, top | are in coordinates of the text drawing window. |
| width, height | size in pixels |

**12.143.4.22 draw_vline()**

```cpp
void Fl_Text_Display::draw_vline (  
    int visLineNum,  
    int leftClip,  
    int rightClip,  
    int leftCharIndex,  
    int rightCharIndex ) [protected]
```

Draw a single line of text. Draw the text on a single line represented by `visLineNum` (the number of lines down from the top of the display), limited by `leftClip` and `rightClip` window coordinates and `leftCharIndex` and `rightCharIndex` character positions (not including the character at position `rightCharIndex`).

Parameters

| visLineNum | index of line in the visible line number lookup |
| leftClip, rightClip | pixel position of clipped area |
| leftCharIndex, rightCharIndex | index into line of segment that we want to draw |

**12.143.4.23 empty_vlines()**

```cpp
int Fl_Text_Display::empty_vlines ( ) const [protected]
```

Return true if there are lines visible with no corresponding buffer text.

Returns

1 if there are empty lines

**12.143.4.24 extend_range_for_styles()**

```cpp
void Fl_Text_Display::extend_range_for_styles (  
    int * startPos,  
    int * endpos ) [protected]
```

I don't know what this does! Extend the range of a redraw request (from `*start` to `*end`) with additional redraw requests resulting from changes to the attached style buffer (which contains auxiliary information for coloring or styling text).

Parameters

| startPos | ?? |
| endpos | ?? |

**Todo** Unicode?
12.143.4.25 *find_line_end()*

```cpp
void Fl_Text_Display::find_line_end (  
    int startPos,  
    bool startPosIsLineStart,  
    int *lineEnd,  
    int *nextLineStart  
) const [protected]
```

Finds both the end of the current line and the start of the next line.

Why? In continuous wrap mode, if you need to know both, figuring out one from the other can be expensive or error prone. The problem comes when there’s a trailing space or tab just before the end of the buffer. To translate an end of line value to or from the next lines start value, you need to know whether the trailing space or tab is being used as a line break or just a normal character, and to find that out would otherwise require counting all the way back to the beginning of the line.

**Parameters**

<table>
<thead>
<tr>
<th>startPos</th>
<th>startPosIsLineStart</th>
<th>lineEnd</th>
<th>nextLineStart</th>
</tr>
</thead>
</table>

12.143.4.26 *find_wrap_range()*

```cpp
void Fl_Text_Display::find_wrap_range (  
    const char *deletedText,  
    int pos,  
    int nInserted,  
    int nDeleted,  
    int *modRangeStart,  
    int *modRangeEnd,  
    int *linesInserted,  
    int *linesDeleted  
) [protected]
```

Wrapping calculations.

When continuous wrap is on, and the user inserts or deletes characters, wrapping can happen before and beyond the changed position. This routine finds the extent of the changes, and counts the deleted and inserted lines over that range. It also attempts to minimize the size of the range to what has to be counted and re-displayed, so the results can be useful both for delimiting where the line starts need to be recalculated, and for deciding what part of the text to redisplay.

**Parameters**

<table>
<thead>
<tr>
<th>deletedText</th>
<th>pos</th>
<th>nInserted</th>
<th>nDeleted</th>
<th>modRangeStart</th>
<th>modRangeEnd</th>
<th>linesInserted</th>
<th>linesDeleted</th>
</tr>
</thead>
</table>

12.143.4.27 *find_x()*

```cpp
int Fl_Text_Display::find_x (  
    const char * s,  
    int len,  
```
Find the index of the character that lies at the given x position / closest cursor position.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>UTF-8 text string</td>
<td></td>
</tr>
<tr>
<td>len</td>
<td>length of string</td>
<td></td>
</tr>
<tr>
<td>style</td>
<td>index into style lookup table</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>position in pixels - negative returns closest cursor position</td>
<td></td>
</tr>
</tbody>
</table>

Returns

index into buffer

12.143.4.28 get_absolute_top_line_number()

int Fl_Text_Display::get_absolute_top_line_number ( ) const [protected]

Returns the absolute (non-wrapped) line number of the first line displayed.
Returns 0 if the absolute top line number is not being maintained.

12.143.4.29 grammar_underline_color() [1/2]

Fl_Color Fl_Text_Display::grammar_underline_color ( ) const [inline]

Gets the underline color for style attribute ATTR_GRAMMAR.

Returns

underline color

12.143.4.30 grammar_underline_color() [2/2]

void Fl_Text_Display::grammar_underline_color ( Fl_Color color ) [inline]

Sets the underline color for style attribute ATTR_GRAMMAR.

Parameters

color underline color

12.143.4.31 handle()

int Fl_Text_Display::handle ( int e ) [virtual]

Event handling.
Reimplemented from Fl_Group.
Reimplemented in Fl_Text_Editor.

12.143.4.32 handle_rmb()

int Fl_Text_Display::handle_rmb ( int readonly ) [protected]

Handle right mouse button down events.
Returns

0 for no op, 1 to cut, 2 to copy, 3 to paste

### 12.143.4.33 handle_vline()

```cpp
int Fl_Text_Display::handle_vline (int mode,
    int lineStartPos,
    int lineLen,
    int leftChar,
    int rightChar,
    int Y,
    int bottomClip,
    int leftClip,
    int rightClip ) const [protected]
```

Universal pixel machine.
We use a single function that handles all line layout, measuring, and drawing

- draw a text range
- return the width of a text range in pixels
- return the index of a character that is at a pixel position

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mode</code></td>
<td>DRAW_LINE, GET_WIDTH, FIND_INDEX, FIND_INDEX_FROM_ZERO, or FIND_CURSOR_INDEX</td>
</tr>
<tr>
<td><code>lineStartPos</code></td>
<td>index of first character</td>
</tr>
<tr>
<td><code>lineLen</code></td>
<td>size of string in bytes</td>
</tr>
<tr>
<td><code>leftChar</code>, <code>rightChar</code></td>
<td>drawing position</td>
</tr>
<tr>
<td><code>Y</code></td>
<td>stop work when we reach the clipped area. rightClip is the X position that we search in FIND_INDEX.</td>
</tr>
<tr>
<td><code>bottomClip</code>, <code>leftClip</code>, <code>rightClip</code></td>
<td>stop work when we reach the clipped area. rightClip is the X position that we search in FIND_INDEX.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAW_LINE</td>
<td>index of last drawn character</td>
</tr>
<tr>
<td>GET_WIDTH</td>
<td>width in pixels of text segment if we would draw it</td>
</tr>
<tr>
<td>FIND_INDEX</td>
<td>index of character at given x position in window coordinates</td>
</tr>
<tr>
<td>FIND_INDEX_FROM_ZERO</td>
<td>index of character at given x position without scrolling and widget offsets</td>
</tr>
</tbody>
</table>

**Todo** we need to handle hidden hyphens and tabs here!
we handle all styles and selections
we must provide code to get pixel positions of the middle of a character as well

### 12.143.4.34 highlight_data()

```cpp
void Fl_Text_Display::highlight_data (Fl_Text_Buffer ∗ styleBuffer,
    const Style_Table_Entry ∗ styleTable,
    int nStyles,
    char unfinishedStyle,
```
Unfinished_Syle_Cb unfinishedHighlightCB,
void * cbArg )

Attach (or remove) highlight information in text display and redisplay.
Highlighting information consists of a style buffer which parallels the normal text buffer, but codes font and color information for the display; a style table which translates style buffer codes (indexed by buffer character - 'A') into fonts and colors; and a callback mechanism for as-needed highlighting, triggered by a style buffer entry of "unfinished←
Style". Style buffer can trigger additional redisplay during a normal buffer modification if the buffer contains a primary Fl_Text_Selection (see extend_range_for_styles() for more information on this protocol).
Style buffers, tables, and their associated memory are managed by the caller.
Styles are ranged from 65 ('A') to 126.

Note

Style information in the style buffer must have the same byte offset as the corresponding character in the text buffer. UTF-8 characters can have a maximum length of four bytes. Style information must take this into account and fill the unused bytes with 0. See fl_utf8len().

Text: "*g r ü r n", where normal style is 'A', and bold is 'B'
Text Buffer(hex): 67 72 c3 bc 6e : gr..n
Style Buffer(hex): 42 41 41 00 42 : BAA.B

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>styleBuffer</td>
<td>this buffer works in parallel to the text buffer. For every character in the text buffer, the style buffer has a byte at the same offset that contains an index into an array of possible styles.</td>
</tr>
<tr>
<td>styleTable</td>
<td>a list of styles indexed by the style buffer</td>
</tr>
<tr>
<td>nStyles</td>
<td>number of styles in the style table</td>
</tr>
<tr>
<td>unfinishedStyle</td>
<td>if this style is found, the callback below is called</td>
</tr>
<tr>
<td>unfinishedHighlightCB</td>
<td>if a character with an unfinished style is found, this callback will be called</td>
</tr>
<tr>
<td>cbArg</td>
<td>an optional argument for the callback above, usually a pointer to the Text Display.</td>
</tr>
</tbody>
</table>

See also

Fl_Text_Display::style_buffer()

12.143.4.35  in_selection()

int Fl_Text_Display::in_selection ( int X, int Y ) const

Check if a pixel position is within the primary selection.

Parameters

| X, Y | pixel position to test |

Returns

1 if position (X, Y) is inside of the primary Fl_Text_Selection

12.143.4.36 insert()

void Fl_Text_Display::insert ( const char * text )

Inserts "text" at the current cursor location.
This has the same effect as inserting the text into the buffer using insert(insert_position(),text) and then moving the insert position after the newly inserted text, except that it's optimized to do less redrawing.
Parameters

text new text in UTF-8 encoding.

12.143.47 insert_position() [1/2]

int Fl_Text_Display::insert_position ( ) const [inline]

Gets the position of the text insertion cursor for text display.
The insert position is the byte count (offset) from the beginning of the text buffer (starting with 0). Returns 0 (zero) if no buffer is associated to the text display. Returns buffer()->length() if the insert position is at the end of the buffer.

Returns

insert position index into text buffer

See also

insert_position(int)

12.143.48 insert_position() [2/2]

void Fl_Text_Display::insert_position ( int newPos )

Sets the position of the text insertion cursor for text display.
Moves the insertion cursor in front of the character at newPos. This function may trigger a redraw.

Parameters

newPos new caret position

12.143.49 line_end()

int Fl_Text_Display::line_end ( int startPos, bool startPosIsLineStart ) const

Returns the end of a line.
Same as buffer()->line_end(startPos), but takes into account line breaks when wrapping is turned on. If the caller knows that startPos is at a line start, it can pass startPosIsLineStart as True to make the call more efficient by avoiding the additional step of scanning back to the last newline.

Note that the definition of the end of a line is less clear when continuous wrap is on. With continuous wrap off, it's just a pointer to the newline that ends the line. When it's on, it's the character beyond the last displayable character on the line, where a whitespace character which has been "converted" to a newline for wrapping is not considered displayable. Also note that a line can be wrapped at a non-whitespace character if the line had no whitespace. In this case, this routine returns a pointer to the start of the next line. This is also consistent with the model used by visLineLength.

Parameters

startPos index to starting character

startPosIsLineStart avoid scanning back to the line start

Returns

new position as index
12.143.4.40  line_start()

int Fl_Text_Display::line_start (  
   int pos ) const

Return the beginning of a line.  
Same as buffer()->line_start(pos), but returns the character after last wrap point rather than the last newline.

Parameters

| pos | index to starting character |

Returns

   new position as index

12.143.4.41  linenumber_align()

void Fl_Text_Display::linenumber_align (  
   Fl_Align val )

Set alignment for line numbers (if enabled).  
Valid values are FL_ALIGN_LEFT, FL_ALIGN_CENTER or FL_ALIGN_RIGHT.

Version

   1.3.3

12.143.4.42  linenumberbgcolor()

void Fl_Text_Display::linenumberbgcolor (  
   Fl_Color val )

Set the background color used for line numbers (if enabled).

Version

   1.3.3

12.143.4.43  linenumberfgcolor()

void Fl_Text_Display::linenumberfgcolor (  
   Fl_Color val )

Set the foreground color used for line numbers (if enabled).

Version

   1.3.3

12.143.4.44  linenumberfont()

void Fl_Text_Display::linenumberfont (  
   Fl_Font val )

Set the font used for line numbers (if enabled).

Version

   1.3.3
### linenumber_format()

```cpp
void Fl_Text_Display::linenumber_format ( const char * val )
```

Sets the printf() style format string used for line numbers. Default is "%d" for normal unpadded decimal integers. An internal copy of `val` is allocated and managed; it is automatically freed whenever a new value is assigned, or when the widget is destroyed. The value of `val` must not be NULL.

Example values:

- "%d" -- For normal line numbers without padding (Default)
- "%03d" -- For 000 padding
- "%x" -- For hexadecimal line numbers
- "%o" -- For octal line numbers

**Version**

1.3.3

### linenumber_size()

```cpp
void Fl_Text_Display::linenumber_size ( Fl_Fontsize val )
```

Set the font size used for line numbers (if enabled).

**Version**

1.3.3

### linenumber_width()

```cpp
void Fl_Text_Display::linenumber_width ( int width )
```

Set width of screen area for line numbers. Use to also enable/disable line numbers. A value of 0 disables line numbering, values >0 enable the line number display.

**Parameters**

| width | The new width of the area for line numbers to appear, in pixels. 0 disables line numbers (default) |

### longest_vline()

```cpp
int Fl_Text_Display::longest_vline ( ) const [protected]
```

Find the longest line of all visible lines.

**Returns**

the width of the longest visible line in pixels

### maintain_absolute_top_line_number()

```cpp
void Fl_Text_Display::maintain_absolute_top_line_number ( int state ) [protected]
```

Line numbering stuff, currently unused.

In continuous wrap mode, internal line numbers are calculated after wrapping. A separate non-wrapped line count is maintained when line numbering is turned on. There is some performance cost to maintaining this line count, so normally absolute line numbers are not tracked if line numbering is off. This routine allows callers to specify that...
they still want this line count maintained (for use via Fl_Text_Display::position_to_linecol()). More specifically, this allows the line number reported in the statistics line to be calibrated in absolute lines, rather than post-wrapped lines.

### 12.143.4.50 maintaining_absolute_top_line_number()

```cpp
int Fl_Text_Display::maintaining_absolute_top_line_number ( ) const [protected]
```

Returns true if a separate absolute top line number is being maintained. The absolute top line number is used for displaying line numbers in continuous wrap mode or showing in the statistics line (the latter is currently not available in FLTK).

### 12.143.4.51 measure_deleted_lines()

```cpp
void Fl_Text_Display::measure_deleted_lines ( int pos, int nDeleted ) [protected]
```

Wrapping calculations. This is a stripped-down version of the findWrapRange() function above, intended to be used to calculate the number of "deleted" lines during a buffer modification. It is called before the modification takes place. This function should only be called in continuous wrap mode with a non-fixed font width. In that case, it is impossible to calculate the number of deleted lines, because the necessary style information is no longer available after the modification. In other cases, we can still perform the calculation afterwards (possibly even more efficiently).

**Parameters**

- `pos`
- `nDeleted`

### 12.143.4.52 measure_proportional_character()

```cpp
double Fl_Text_Display::measure_proportional_character ( const char * s, int xPix, int pos ) const [protected]
```

Wrapping calculations. Measure the width in pixels of the first character of string "s" at a particular column "colNum" and buffer position "pos". This is for measuring characters in proportional or mixed-width highlighting fonts.

A note about proportional and mixed-width fonts: the mixed width and proportional font code in nedit does not get much use in general editing, because nedit doesn't allow per-language-mode fonts, and editing programs in a proportional font is usually a bad idea, so very few users would choose a proportional font as a default. There are still probably mixed-width syntax highlighting cases where things don't redraw properly for insertion/deletion, though static display and wrapping and resizing should now be solid because they are now used for online help display.

**Parameters**

- `s` text string
- `xPix` x pixel position needed for calculating tab widths
- `pos` offset within string
Returns
width of character in pixels

12.143.4.53  measure_vline()

int Fl_Text_Display::measure_vline {
    int visLineNum) const  [protected]

Returns the width in pixels of the displayed line pointed to by "visLineNum".

Parameters
visLineNum  index into visible lines array

Returns
width of line in pixels

12.143.4.54  move_down()

int Fl_Text_Display::move_down ( )

Moves the current insert position down one line.

Returns
1 if the cursor moved, 0 if the beginning of the text was reached

12.143.4.55  move_left()

int Fl_Text_Display::move_left ( )

Moves the current insert position left one character.

Returns
1 if the cursor moved, 0 if the beginning of the text was reached

12.143.4.56  move_right()

int Fl_Text_Display::move_right ( )

Moves the current insert position right one character.

Returns
1 if the cursor moved, 0 if the end of the text was reached

12.143.4.57  move_up()

int Fl_Text_Display::move_up ( )

Moves the current insert position up one line.

Returns
1 if the cursor moved, 0 if the beginning of the text was reached

12.143.4.58  offset_line_starts()

void Fl_Text_Display::offset_line_starts {
    int newTopLineNum) [protected]

Offset line start counters for a new vertical scroll position.
Offset the line starts array, mTopLineNum, mFirstChar and lastChar, for a new vertical scroll position given by newTopLineNum. If any currently displayed lines will still be visible, salvage the line starts values, otherwise, count lines from the nearest known line start (start or end of buffer, or the closest value in the mLineStarts array)
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>newTopLineNum</code></td>
<td>index into buffer</td>
</tr>
</tbody>
</table>

12.143.459  **overstrike()**

```cpp
void Fl_Text_Display::overstrike (const char * text)
```

Replaces text at the current insert position.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>text</code></td>
<td>new text in UTF-8 encoding</td>
</tr>
</tbody>
</table>

**Todo**  Unicode? Find out exactly what we do here and simplify.

12.143.460  **position_style()**

```cpp
int Fl_Text_Display::position_style (int lineStartPos, int lineLen, int lineIndex) const
```

Find the correct style for a character.

Determine the drawing method to use to draw a specific character from "buf". `lineStartPos` gives the character index where the line begins, `lineIndex`, the number of characters past the beginning of the line, and `lineLen` the number of displayed characters past the beginning of the line. Passing `lineStartPos` of -1 returns the drawing style for "no text".

Why not just: `position_style(pos)`? Because style applies to blank areas of the window beyond the text boundaries, and because this routine must also decide whether a position is inside of a rectangular `Fl_Text_Selection`, and do so efficiently, without re-counting character positions from the start of the line.

Note that style is a somewhat incorrect name, drawing method would be more appropriate.

If `lineIndex` is pointing to the last character in a line, and the second to last character has the `ATTR_BGCOLOR_EXT` set, the background color will extend into the remaining line.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lineStartPos</code></td>
<td>beginning of this line</td>
</tr>
<tr>
<td><code>lineLen</code></td>
<td>number of bytes in line</td>
</tr>
<tr>
<td><code>lineIndex</code></td>
<td>position of character within line</td>
</tr>
</tbody>
</table>

Returns

- style for the given character

12.143.461  **position_to_line()**

```cpp
int Fl_Text_Display::position_to_line (int pos, int * lineNum) const [protected]
```

Convert a position index into a line number offset.

Find the line number of position `pos` relative to the first line of displayed text, counting from 0 to `visible lines - 1`. The line number is returned in `lineNum`.

Returns 0 if the line is not displayed. In this case `lineNum` is 0 as well.

Returns 1 if the line is displayed. In this case `lineNum` is the relative line number.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>pos</th>
<th>byte position in buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>lineNum</td>
<td>relative line number of byte pos in buffer</td>
</tr>
</tbody>
</table>

Returns

whether the character at byte position pos is currently displayed

Return values

| 0 | pos is not displayed; lineNum is invalid (zero) |
| 1 | pos is displayed; lineNum is valid |

12.143.4.62 position_to_linecol()

int Fl_Text_Display::position_to_linecol (  
    int pos,  
    int * lineNum,  
    int * column ) const [protected]

Find the line and column number of position pos.
This only works for displayed lines. If the line is not displayed, the function returns 0 (without the mLineStarts array it could turn in to very long calculation involving scanning large amounts of text in the buffer). If continuous wrap mode is on, returns the absolute line number (as opposed to the wrapped line number which is used for scrolling).

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>character index</th>
</tr>
</thead>
<tbody>
<tr>
<td>lineNum</td>
<td>absolute (unwrapped) line number</td>
</tr>
<tr>
<td>column</td>
<td>character offset to the beginning of the line</td>
</tr>
</tbody>
</table>

Returns

0 if pos is off screen, line number otherwise

Todo a column number makes little sense in the UTF-8/variable font width environment. We will have to further define what exactly we want to return. Please check the functions that call this particular function.

12.143.4.63 position_to_xy()

int Fl_Text_Display::position_to_xy (  
    int pos,  
    int * X,  
    int * Y ) const

Convert a character index into a pixel position.
Translate a buffer text position to the XY location where the top left of the cursor would be positioned to point to that character. Returns 0 if the position is not displayed because it is vertically out of view. If the position is horizontally out of view, returns the X coordinate where the position would be if it were visible.

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>character index</th>
</tr>
</thead>
<tbody>
<tr>
<td>X, Y</td>
<td>pixel position of character on screen</td>
</tr>
</tbody>
</table>
Returns

0 if character vertically out of view, X & Y positions otherwise

12.143.4.64  redisplay_range()

```cpp
void Fl_Text_Display::redisplay_range (int startpos, int endpos)
```

Marks text from start to end as needing a redraw.
This function will trigger a damage event and later a redraw of parts of the widget.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startpos</td>
<td>index of first character needing redraw</td>
</tr>
<tr>
<td>endpos</td>
<td>index after last character needing redraw</td>
</tr>
</tbody>
</table>

12.143.4.65  reset_absolute_top_line_number()

```cpp
void Fl_Text_Display::reset_absolute_top_line_number ( ) [protected]
```

Reestablish the absolute (non-wrapped) top line number.
Count lines from the beginning of the buffer to reestablish the absolute (non-wrapped) top line number. If mode is not continuous wrap, or the number is not being maintained, does nothing.

12.143.4.66  resize()

```cpp
void Fl_Text_Display::resize (int X, int Y, int W, int H) [virtual]
```

Change the size of the displayed text area.
Calling this function will trigger a recalculation of all visible lines and of all scrollbar sizes.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X,Y,W,H</td>
<td>new position and size of this widget</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Group.

12.143.4.67  rewind_lines()

```cpp
int Fl_Text_Display::rewind_lines (int startPos, int nLines)
```

Skip a number of lines back.
Same as buffer()->rewind_lines(startPos, nLines), but takes into account line breaks when wrapping is turned on.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startPos</td>
<td>index to starting character</td>
</tr>
<tr>
<td>nLines</td>
<td>number of lines to skip back</td>
</tr>
</tbody>
</table>
12.143.4.68 scroll()

void Fl_Text_Display::scroll ( 
    int topLineNum, 
    int horizOffset )

Scrolls the current buffer to start at the specified line and column.

Parameters

<table>
<thead>
<tr>
<th>topLineNum</th>
<th>top line number</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizOffset</td>
<td>column number</td>
</tr>
</tbody>
</table>

**Todo** Column numbers make little sense here.

12.143.4.69 scroll()

int Fl_Text_Display::scroll_ ( 
    int topLineNum, 
    int horizOffset ) [protected]

Scrolls the current buffer to start at the specified line and column.

Parameters

<table>
<thead>
<tr>
<th>topLineNum</th>
<th>top line number</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizOffset</td>
<td>in pixels</td>
</tr>
</tbody>
</table>

Returns

0 if nothing changed, 1 if we scrolled

12.143.4.70 scroll_timer_cb()

void Fl_Text_Display::scroll_timer_cb ( 
    void * user_data ) [static], [protected]

Timer callback for scroll events.
This timer event scrolls the text view proportionally to how far the mouse pointer has left the text area. This allows for smooth scrolling without "wiggeling" the mouse.

12.143.4.71 scrollbar_align()[1/2]

Fl_Align Fl_Text_Display::scrollbar_align ( ) const [inline]

Gets the scrollbar alignment type.

Returns

scrollbar alignment

12.143.4.72 scrollbar_align()[2/2]

void Fl_Text_Display::scrollbar_align ( 
    Fl_Align a ) [inline]

Sets the scrollbar alignment type.
Parameters

\begin{verbatim}
a new scrollbar alignment
\end{verbatim}

12.143.4.73 scrollbar_size() [1/2]

```
int Fl_Text_Display::scrollbar_size () const [inline]
```

Gets the current size of the scrollbars' troughs, in pixels.
If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.

Returns

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int)

12.143.4.74 scrollbar_size() [2/2]

```
void Fl_Text_Display::scrollbar_size (
   int newSize ) [inline]
```

Sets the pixel size of the scrollbars' troughs to newSize, in pixels.
Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.
Only use THIS method if you really need to override the global scrollbar size. Setting newSize to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.

Parameters

\begin{verbatim}
in newSize Sets the scrollbar size in pixels.
If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()
\end{verbatim}

See also

Fl::scrollbar_size()

12.143.4.75 scrollbar_width() [1/2]

```
int Fl_Text_Display::scrollbar_width () const [inline]
```

Returns the global value Fl::scrollbar_size() unless a specific scrollbar_width_ has been set.

Deprecated Use scrollbar_size() instead.

Todo This method should eventually be removed.

12.143.4.76 scrollbar_width() [2/2]

```
void Fl_Text_Display::scrollbar_width ( 
   int width ) [inline]
```

Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

Deprecated Use scrollbar_size() instead.

Todo This method should eventually be removed
12.143.4.77  secondary_selection_color() [1/2]

**Fl_Color Fl_Text_Display::secondary_selection_color ( ) const [inline]**

Gets the background color for the secondary selection block.

Returns

- background color

12.143.4.78  secondary_selection_color() [2/2]

**void Fl_Text_Display::secondary_selection_color ( Fl_Color color ) [inline]**

Sets the background color for the secondary selection block.

Parameters

- color  background color

12.143.4.79  shortcut() [1/2]

**int Fl_Text_Display::shortcut ( ) const [inline]**

**Todo** FIXME : get set methods pointing on shortcut_ have no effects as shortcut_ is unused in this class and derived!

Returns

- the current shortcut key

12.143.4.80  shortcut() [2/2]

**void Fl_Text_Display::shortcut ( int s ) [inline]**

**Todo** FIXME : get set methods pointing on shortcut_ have no effects as shortcut_ is unused in this class and derived!

Parameters

- s  the new shortcut key

12.143.4.81  show_cursor()

**void Fl_Text_Display::show_cursor ( int b = 1 )**

Shows the text cursor.

This function may trigger a redraw.

Parameters

- b  show(1) or hide(0) the text cursor (caret).
12.143.4.82  show_insert_position()

void Fl_Text_Display::show_insert_position ( )
Scrolls the text buffer to show the current insert position.
This function triggers a complete recalculation, ending in a call to Fl_Text_Display::display_insert()

12.143.4.83  skip_lines()

int Fl_Text_Display::skip_lines ( int startPos, int nLines, bool startPosIsLineStart )
Skip a number of lines forward.
Same as Fl_Text_Buffer::skip_lines(startPos, nLines), but takes into account line breaks when wrapping is turned on. If the caller knows that startPos is at a line start, it can pass startPosIsLineStart as True to make the call more efficient by avoiding the additional step of scanning back to the last newline.

Parameters

<table>
<thead>
<tr>
<th>startPos</th>
<th>index to starting character</th>
</tr>
</thead>
<tbody>
<tr>
<td>nLines</td>
<td>number of lines to skip ahead</td>
</tr>
<tr>
<td>startPosIsLineStart</td>
<td>avoid scanning back to the line start</td>
</tr>
</tbody>
</table>

Returns

new position as index

12.143.4.84  spelling_underline_color() [1/2]

Fl_Color Fl_Text_Display::spelling_underline_color { } const [inline]
Gets the underline color for style attribute ATTR_SPELLING.

Returns

underline color

12.143.4.85  spelling_underline_color() [2/2]

void Fl_Text_Display::spelling_underline_color ( Fl_Color color ) [inline]
Sets the underline color for style attribute ATTR_SPELLING.

Parameters

| color | underline color |

12.143.4.86  string_width()

double Fl_Text_Display::string_width ( const char * string, int length, int style ) const [protected]
Find the width of a string in the font of a particular style.

Parameters

| string | the text |
Parameters

<table>
<thead>
<tr>
<th>length</th>
<th>number of bytes in string</th>
</tr>
</thead>
<tbody>
<tr>
<td>style</td>
<td>index into style table</td>
</tr>
</tbody>
</table>

Returns

width of text segment in pixels

12.143.4.87  style_buffer()

*Fl_Text_Buffer* Fl_Text_Display::style_buffer() const [inline]

Gets the current style buffer associated with the text widget.
Multiple text widgets can be associated with the same style buffer.

Returns

current style buffer

See also

Fl_Text_Display::highlight_data()

12.143.4.88  textcolor() [1/2]

*Fl_Color* Fl_Text_Display::textcolor() const [inline]

Gets the default color of text in the widget.

Returns

text color unless overridden by a style

12.143.4.89  textcolor() [2/2]

void Fl_Text_Display::textcolor(
    *Fl_Color* n ) [inline]

Sets the default color of text in the widget.

Parameters

| n       | new text color |

12.143.4.90  textfont() [1/2]

*Fl_Font* Fl_Text_Display::textfont() const [inline]

Gets the default font used when drawing text in the widget.

Returns

current text font face unless overridden by a style

12.143.4.91  textfont() [2/2]

void Fl_Text_Display::textfont(
    *Fl_Font* s ) [inline]

Sets the default font used when drawing text in the widget.
Parameters

s default text font face

12.143.4.92  textbox() [1/2]

Fl_Fontsize Fl_Text_Display::textsize ( ) const [inline]
Gets the default size of text in the widget.

Returns

current text height unless overridden by a style

12.143.4.93  textbox() [2/2]

void Fl_Text_Display::textbox ( Fl_Fontsize s ) [inline]
Sets the default size of text in the widget.

Parameters

s new text size

12.143.4.94  update_h_scrollbar()

void Fl_Text_Display::update_h_scrollbar ( ) [protected]
Update horizontal scrollbar.
Update the minimum, maximum, slider size, page increment, and value for the horizontal scrollbar.

12.143.4.95  update_line_starts()

void Fl_Text_Display::update_line_starts ( int pos,
int charsInserted,
int charsDeleted,
int linesInserted,
int linesDeleted,
int * scrolled ) [protected]
Update line start arrays and variables.
Update the line starts array, mTopLineNum, mFirstChar and lastChar for this text display after a modification to the text buffer, given by the position pos where the change began, and the numbers of characters and lines inserted and deleted.

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>index into buffer of recent changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>charsInserted</td>
<td>number of bytes(!) inserted</td>
</tr>
<tr>
<td>charsDeleted</td>
<td>number of bytes(!) deleted</td>
</tr>
<tr>
<td>linesInserted</td>
<td>number of lines</td>
</tr>
<tr>
<td>linesDeleted</td>
<td>number of lines</td>
</tr>
<tr>
<td>out</td>
<td>scrolled set to 1 if the text display needs to be scrolled</td>
</tr>
</tbody>
</table>

12.143.4.96  update_v_scrollbar()

void Fl_Text_Display::update_v_scrollbar ( ) [protected]

Generated by Doxygen
Update vertical scrollbar.
Update the minimum, maximum, slider size, page increment, and value for the vertical scrollbar.

12.143.4.97  vline_length()

```cpp
text
int Fl_Text_Display::vline_length (int visLineNum) const [protected]
```

Count number of bytes in a visible line.
Return the length of a line (number of bytes) by examining entries in the line starts array rather than by scanning for newlines.

Parameters

| visLineNum | index of line in visible line array |

Returns

text

number of bytes in this line

12.143.4.98  word_end()

```cpp
text
int Fl_Text_Display::word_end (int pos) const [inline]
```

Moves the insert position to the end of the current word.

Parameters

| pos | start calculation at this index |

Returns

text

index of first character after the end of the word

12.143.4.99  word_start()

```cpp
text
int Fl_Text_Display::word_start (int pos) const [inline]
```

Moves the insert position to the beginning of the current word.

Parameters

| pos | start calculation at this index |

Returns

text

beginning of the words

12.143.4.100  wrap_mode()

```cpp
text
void Fl_Text_Display::wrap_mode (int wrap, int wrapMargin)
```

Set the new text wrap mode.
If wrap mode is not zero, this call enables automatic word wrapping at column wrapMargin. Word-wrapping does not change the text buffer itself, only the way the text is displayed. Different Text Displays can have different wrap modes, even if they share the same Text Buffer.
Valid wrap modes are:

- WRAP_NONE: don't wrap text at all
- WRAP_AT_COLUMN: wrap text at the given text column
- WRAP_AT_PIXEL: wrap text at a pixel position
- WRAP_AT_BOUNDS: wrap text so that it fits into the widget width

Parameters

<table>
<thead>
<tr>
<th>wrap</th>
<th>new wrap mode (see above)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrapMargin</td>
<td>In WRAP_AT_COLUMN mode, text will wrap at the n'th character. For variable width fonts, an average character width is calculated. The column width is calculated using the current textfont or the first style when this function is called. If the font size changes, this function must be called again. In WRAP_AT_PIXEL mode, this is the pixel position.</td>
</tr>
</tbody>
</table>

12.143.4.101 wrap Uses_character()

int Fl_Text_Display::wrap_uses_character (int lineEndPos) const [protected]

Check if the line break is caused by a newline or by line wrapping.

Line breaks in continuous wrap mode usually happen at newlines ('n') or whitespace. This line-terminating character is not included in line width measurements and has a special status as a non-visible character. However, lines with no whitespace are wrapped without the benefit of a line terminating character, and this distinction causes endless trouble with all of the text display code which was originally written without continuous wrap mode and always expects to wrap at a newline character.

Given the position of the end of the line, as returned by Fl_Text_Display::line_end() or Fl_Text_Buffer::line_end(), this returns true if there is a line terminating character, and false if there's not. On the last character in the buffer, this function can't tell for certain whether a trailing space was used as a wrap point, and just guesses that it wasn't. So if an exact accounting is necessary, don't use this function.

Parameters

| lineEndPos | index of character where the line wraps |

Returns

1 if a \n character causes the line wrap

12.143.4.102 wrapped_column()

int Fl_Text_Display::wrapped_column (int row, int column) const

Nobody knows what this function does.

Correct a column number based on an unconstrained position (as returned by TextDXYToUnconstrainedPosition) to be relative to the last actual newline in the buffer before the row and column position given, rather than the last line start created by line wrapping. This is an adapter for rectangular selections and code written before continuous wrap mode, which thinks that the unconstrained column is the number of characters from the last newline. Obviously this is time consuming, because it involves character re-counting.

Parameters

| row | |
| column | |
Returns

something unknown

Todo  What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)

Todo  Unicode?

12.143.4.103  wrapped_line_counter()

void Fl_Text_Display::wrapped_line_counter (  
    Fl_Text_Buffer * buf,  
    int startPos,  
    int maxPos,  
    int maxLines,  
    bool startPosIsLineStart,  
    int styleBufOffset,  
    int * retPos,  
    int * retLines,  
    int * retLineStart,  
    int * retLineEnd,  
    bool countLastLineMissingNewLine = true ) const [protected]

Wrapping calculations.
Count forward from startPos to either maxPos or maxLines (whichever is reached first), and return all relevant positions and line count. The provided textBuffer may differ from the actual text buffer of the widget. In that case it must be a (partial) copy of the actual text buffer and the styleBufOffset argument must indicate the starting position of the copy, to take into account the correct style information.

Parameters

| in   | buf                        | The text buffer to operate on |
| in   | startPos                   | Starting index position into the buffer |
| in   | maxPos                     | Maximum index position into the buffer we'll reach |
| in   | maxLines                   | Maximum number of lines we'll reach |
| in   | startPosIsLineStart        | Flag indicating if startPos is start of line. (If set, prevents our having to find the line start) |
| in   | styleBufOffset             | Offset index position into style buffer. |
| out  | retPos                     | Position where counting ended. When counting lines, the position returned is the start of the line "maxLines" lines beyond "startPos". |
| out  | retLines                   | Number of line breaks counted |
| out  | retLineStart               | Start of the line where counting ended |
| out  | retLineEnd                 | End position of the last line traversed |
| out  | countLastLineMissingNewLine | |

12.143.4.104  wrapped_row()

int Fl_Text_Display::wrapped_row (  
    int row ) const

Nobody knows what this function does.
Correct a row number from an unconstrained position (as returned by TextDXYToUnconstrainedPosition) to a straight number of newlines from the top line of the display. Because rectangular selections are based on newlines, rather than display wrapping, and anywhere a rectangular selection needs a row, it needs it in terms of un-wrapped lines.
**Parameters**

- **row**

**Returns**

- something unknown

**Todo** What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)

### x_to_col()

```cpp
double Fl_Text_Display::x_to_col ( double x ) const
```

Convert an x pixel position into a column number.

**Parameters**

- **x** number of pixels from the left margin

**Returns**

- an approximate column number based on the main font

### xy_to_position()

```cpp
int Fl_Text_Display::xy_to_position ( int X, int Y, int posType = CHARACTER_POS ) const [protected]
```

Translate a pixel position into a character index.

Translate window coordinates to the nearest (insert cursor or character cell) text position. The parameter `posType` specifies how to interpret the position: CURSOR_POS means translate the coordinates to the nearest cursor position, and CHARACTER_POS means return the position of the character closest to \((X, Y)\).

**Parameters**

<table>
<thead>
<tr>
<th>X, Y</th>
<th>pixel position</th>
</tr>
</thead>
<tbody>
<tr>
<td>posType</td>
<td>CURSOR_POS or CHARACTER_POS</td>
</tr>
</tbody>
</table>

**Returns**

- index into text buffer

### xy_to_rowcol()

```cpp
void Fl_Text_Display::xy_to_rowcol ( int X, int Y, int * row, int * column, int posType = CHARACTER_POS ) const [protected]
```

Translate pixel coordinates into row and column.

Translate window coordinates to the nearest row and column number for positioning the cursor. This, of course, makes no sense when the font is proportional, since there are no absolute columns. The parameter `posType`
specifies how to interpret the position: CURSOR_POS means translate the coordinates to the nearest position between characters, and CHARACTER_POS means translate the position to the nearest character cell.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X,Y</td>
<td>pixel coordinates</td>
</tr>
<tr>
<td>out</td>
<td>row, column nearest row and column</td>
</tr>
<tr>
<td>posType</td>
<td>CURSOR_POS or CHARACTER_POS</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Text_Display.H
- Fl_Text_Display.cxx

**12.144 Fl_Text_Editor Class Reference**

This is the FLTK text editor widget.

```cpp
#include <Fl_Text_Editor.H>
```

Inheritance diagram for Fl_Text_Editor:

```
Fl_Widget
   ↓
Fl_Group
   ↓
Fl_Text_Display
   ↓
Fl_Text_Editor
```

Classes

- struct **Key_Binding**
  
  Simple linked list item associating a key/state to a function.

Public Types

- typedef int(∗Key_Func)(int key, Fl_Text_Editor ∗editor)
  
  Key function binding callback type.

Public Types inherited from Fl_Text_Display

- enum {NORMAL_CURSOR, CARET_CURSOR, DIM_CURSOR, BLOCK_CURSOR, HEAVY_CURSOR, SIMPLE_CURSOR}
  
  text display cursor shapes enumeration
- enum {CURSOR_POS, CHARACTER_POS}
  
  the character position is the left edge of a character, whereas the cursor is thought to be between the centers of two consecutive characters.
- enum {
  DRAG_NONE = -2, DRAG_START_DND = -1, DRAG_CHAR = 0, DRAG_WORD = 1, DRAG_LINE = 2
  }
  
  drag types - they match Fl::event_clicks() so that single clicking to start a collection selects by character, double clicking selects by word and triple clicking selects by line.
- enum {WRAP_NONE, WRAP_AT_COLUMN, WRAP_AT_PIXEL, WRAP_AT_BOUNDS}
wrap types - used in wrap_mode()

• enum {
  ATTR_BGCOLOR = 0x0001, ATTR_BGCOLOR_EXT = 0x0002, ATTR_BGCOLOR_EXT = 0x0003,
  ATTR_UNDERLINE = 0x0004,
  ATTR_GRAMMAR = 0x0008, ATTR_SPELLING = 0x000C, ATTR_STRIKE THROUGH = 0x0010,
  ATTR_LINES_MASK = 0x001C
}

attribute flags in Style_Table_Entry.attr

• typedef void(* Unfinished_Style_Cb) (int, void *)

Public Member Functions

• void add_default_key_bindings (Key_Binding **list)
  Adds all of the default editor key bindings to the specified key binding list.

• void add_key_binding (int key, int state, Key_Func f)
  Adds a key of state state with the function f.

• void add_key_binding (int key, int state, Key_Func f, Key_Binding **list)
  Adds a key of state state with the function function to an arbitrary key binding list list.

• Key_Func bound_key_function (int key, int state) const
  Returns the function associated with a key binding.

• Key_Func bound_key_function (int key, int state, Key_Binding *list) const
  Returns the function associated with a key binding.

• void default_key_function (Key_Func f)
  Sets the default key function for unassigned keys.

• Fl_Text_Editor (int X, int Y, int W, int H, const char *l=0)
  The constructor creates a new text editor widget.

• int handle (int e) FL_OVERRIDE
  Event handling.

• int insert_mode ()
  Gets the current insert mode; if non-zero, new text is inserted before the current cursor position.

• void insert_mode (int b)
  Sets the current insert mode; if non-zero, new text is inserted before the current cursor position.

• void remove_all_key_bindings ()
  Removes all of the key bindings associated with the text editor or list.

• void remove_all_key_bindings (Key_Binding **list)
  Removes all of the key bindings associated with the text editor or list.

• void remove_key_binding (int key, int state)
  Removes the key binding associated with the key "key" of state "state".

• void remove_key_binding (int key, int state, Key_Binding **list)
  Removes the key binding associated with the key key of state state from the Key_Binding list list.

• int tab_nav () const
  Checks if Tab focus navigation is enabled.

• void tab_nav (int val)
  Enables or disables Tab key focus navigation.

Public Member Functions inherited from Fl_Text_Display

• Fl_Text_Buffer * buffer () const
  Gets the current text buffer associated with the text widget.

• void buffer (Fl_Text_Buffer &buf)
  Sets the current text buffer associated with the text widget.

• void buffer (Fl_Text_Buffer +buf)
  Attach a text buffer to display, replacing the current buffer (if any).
• double col_to_x (double col) const
  Convert a column number into an x pixel position.
• int count_lines (int start, int end, bool start_pos_is_line_start) const
  Count the number of lines between two positions.
• Fl_Color cursor_color () const
  Gets the text cursor color.
• void cursor_color (Fl_Color n)
  Sets the text cursor color.
• int cursor_style () const
• void cursor_style (int style)
  Sets the text cursor style.
• Fl_Text_Display (int X, int Y, int W, int H, const char ∗l=0)
  Creates a new text display widget.
• Fl_Color grammar_underline_color () const
  Gets the underline color for style attribute ATTR_GRAMMAR.
• void grammar_underline_color (Fl_Color color)
  Sets the underline color for style attribute ATTR_GRAMMAR.
• void hide_cursor ()
  Hides the text cursor.
• void highlight_data (Fl_Text_Buffer ∗styleBuffer, const Style_Table_Entry ∗styleTable, int nStyles, char unfinishedStyle, Unfinished_Style_Cb unfinishedHighlightCB, void ∗cbArg)
  Attach (or remove) highlight information in text display and redisplay.
• int in_selection (int x, int y) const
  Check if a pixel position is within the primary selection.
• void insert (const char ∗text)
  Inserts "text" at the current cursor location.
• int insert_position () const
  Gets the position of the text insertion cursor for text display.
• void insert_position (int newPos)
  Sets the position of the text insertion cursor for text display.
• int line_end (int startPos, bool startPosIsLineStart) const
  Returns the end of a line.
• int line_start (int pos) const
  Return the beginning of a line.
• Fl_Align linenumber_align () const
  Returns the alignment used for line numbers (if enabled).
• void linenumber_align (Fl_Align val)
  Set alignment for line numbers (if enabled).
• Fl_Color linenumberbgcolor () const
  Returns the background color used for line numbers (if enabled).
• void linenumberbgcolor (Fl_Color val)
  Set the background color used for line numbers (if enabled).
• Fl_Color linenumberfgcolor () const
  Return the foreground color used for line numbers (if enabled).
• void linenumberfgcolor (Fl_Color val)
  Set the foreground color used for line numbers (if enabled).
• Fl_Font linenumberfont () const
  Return the font used for line numbers (if enabled).
• void linenumberfont (Fl_Font val)
  Set the font used for line numbers (if enabled).
• const char ∗linenumberformat () const
Returns the line number printf() format string.

- void linenumber_format (const char *val)
  Sets the printf() style format string used for line numbers.

- Fl_Fontsize linenumber_size () const
  Return the font size used for line numbers (if enabled).

- void linenumber_size (Fl_Fontsize val)
  Set the font size used for line numbers (if enabled).

- int linenumber_width () const
  Return the screen area width provided for line numbers.

- void linenumber_width (int width)
  Set width of screen area for line numbers.

- int move_down ()
  Moves the current insert position down one line.

- int move_left ()
  Moves the current insert position left one character.

- int move_right ()
  Moves the current insert position right one character.

- int move_up ()
  Moves the current insert position up one line.

- void next_word (void)
  Moves the current insert position right one word.

- void overstrike (const char *text)
  Replaces text at the current insert position.

- int position_style (int lineStartPos, int lineLen, int lineIndex) const
  Find the correct style for a character.

- int position_to_xy (int pos, int *x, int *y) const
  Convert a character index into a pixel position.

- void previous_word (void)
  Moves the current insert position left one word.

- virtual void recalc_display ()
  Recalculate the display’s visible lines and scrollbar sizes.

- void redisplay_range (int start, int end)
  Marks text from start to end as needing a redraw.

- void resize (int X, int Y, int W, int H) FL_OVERRIDE
  Change the size of the displayed text area.

- int rewind_lines (int startPos, int nLines)
  Skip a number of lines back.

- void scroll (int topLineNum, int horizOffset)
  Scrolls the current buffer to start at the specified line and column.

- Fl_Align scrollbar_align () const
  Gets the scrollbar alignment type.

- void scrollbar_align (Fl_Align a)
  Sets the scrollbar alignment type.

- int scrollbar_size () const
  Gets the current size of the scrollbars' troughs, in pixels.

- void scrollbar_size (int newSize)
  Sets the pixel size of the scrollbars' troughs to newSize, in pixels.

- int scrollbar_width () const
  Returns the global value Fl::scrollbar_size() unless a specific scrollbar_width_ has been set.

- void scrollbar_width (int width)
  Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.
- **Fl_Color secondary_selection_color () const**
  
  Gets the background color for the secondary selection block.

- **void secondary_selection_color (Fl_Color color)**
  
  Sets the background color for the secondary selection block.

- **int shortcut () const**

- **void shortcut (int s)**

- **void show_cursor (int b=1)**
  
  Shows the text cursor.

- **void show_insert_position ()**
  
  Scrolls the text buffer to show the current insert position.

- **int skip_lines (int startPos, int nLines, bool startPosIsLineStart)**
  
  Skip a number of lines forward.

- **Fl_Color spelling_underline_color () const**
  
  Gets the underline color for style attribute ATTR_SPELLING.

- **void spelling_underline_color (Fl_Color color)**
  
  Sets the underline color for style attribute ATTR_SPELLING.

- **Fl_Text_Buffer * style_buffer () const**
  
  Gets the current style buffer associated with the text widget.

- **Fl_Color textcolor () const**
  
  Gets the default color of text in the widget.

- **void textcolor (Fl_Color n)**
  
  Sets the default color of text in the widget.

- **Fl_Font textfont () const**
  
  Gets the default font used when drawing text in the widget.

- **void textfont (Fl_Font s)**
  
  Sets the default font used when drawing text in the widget.

- **Fl_Fontsize textsize () const**
  
  Gets the default size of text in the widget.

- **void textsize (Fl_Fontsize s)**
  
  Sets the default size of text in the widget.

- **int word_end (int pos) const**
  
  Moves the insert position to the end of the current word.

- **int word_start (int pos) const**
  
  Moves the insert position to the beginning of the current word.

- **void wrap_mode (int wrap, int wrap_margin)**
  
  Set the new text wrap mode.

- **int wrapped_column (int row, int column) const**
  
  Nobody knows what this function does.

- **int wrapped_row (int row) const**
  
  Nobody knows what this function does.

- **double x_to_col (double x) const**
  
  Convert an x pixel position into a column number.

- **Fl_Text_Display ()**
  
  Free a text display and release its associated memory.
Public Member Functions inherited from Fl_Group

- **Fi_Widget** & _ddfdesign_kludge ()
  This is for forms compatibility only.
- void add (Fl_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.
- void add (Fl_Widget *o)
  See void Fl_Group::add(Fl_Widget &w)
- void add_resizable (Fl_Widget &o)
  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const
  Returns a pointer to the array of children.
- Fl_Group *as_group () const FL_OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()
  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const
  Returns array()[n].
- int children () const
  Returns how many child widgets the group has.
- void clear ()
  Deletes all child widgets from memory recursively.
- unsigned int clip_children ()
  Returns the current clipping mode.
- void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.
- void end ()
  Exactly the same as current(this->parent()).
- int find (const Fl_Widget &o) const
  See int Fl_Group::find(const Fl_Widget *w) const.
- int find (const Fl_Widget *) const
  Searches the child array for the widget and returns the index.
- Fl_Group (int, int, int, int, const char * =0)
  Creates a new Fl_Group widget using the given position, size, and label string.
- void focus (Fl_Widget *W)
- void forms_end ()
  This is for forms compatibility only.
- void init_sizes ()
  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)
  This does insert(w, find(before)).
- void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)
  Removes the widget o from the group.
- void remove (int index)
Removes the widget at index from the group but does not delete it.

- **Fl_Widget** ∗ resizable () const
  Returns the group's resizable widget.

- **void** resizable (**Fl_Widget** &o)
  Sets the group's resizable widget.

- **void** resizable (**Fl_Widget** ∗o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- **virtual** ~**Fl_Group** ()
  The destructor also deletes all the children.

### Public Member Functions inherited from **Fl_Widget**

- **void** _clear_fullscreen ()
- **void** _set_fullscreen ()
- **void** activate ()
  Activates the widget.

- **unsigned int** active () const
  Returns whether the widget is active.

- **int** active_r () const
  Returns whether the widget and all of its parents are active.

- **Fl_Align** align () const
  Gets the label alignment.

- **void** align (**Fl_Align** alignment)
  Sets the label alignment.

- **long** argument () const
  Gets the current user data (long) argument that is passed to the callback function.

- **void** argument (**long** v)
  Sets the current user data (long) argument that is passed to the callback function.

- **virtual** class **Fl_Gl_Window** ∗ as_gl_window ()
  Returns an **Fl_Gl_Window** pointer if this widget is an **Fl_Gl_Window**.

- **virtual** class **Fl_Gl_Window** const ∗ as_gl_window () const
- **virtual** class **Fl_Window** ∗ as_window ()
  Returns an **Fl_Window** pointer if this widget is an **Fl_Window**.

- **virtual** **Fl_Window** const ∗ as_window () const
- **void** bind_deimage (**Fl_Image** ∗img)
  Sets the image to use as part of the widget label when in the inactive state.

- **void** bind_deimage (**int** f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- **void** bind_image (**Fl_Image** ∗img)
  Sets the image to use as part of the widget label when in the active state.

- **void** bind_image (**int** f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype** box () const
  Gets the box type of the widget.

- **void** box (**Fl_Boxtype** new_box)
  Sets the box type for the widget.

- **Fl_Callback_p** callback () const
  Gets the current callback function for the widget.

- **void** callback (**Fl_Callback** ∗cb)
  Sets the current callback function for the widget.

- **void** callback (**Fl_Callback** ∗cb, **Fl_Callback_User_Data** ∗p, **bool** auto_free)
Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.

- void callback (Fl_Callback0 *cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.

- unsigned int changed () const
  Checks if the widget value changed since the last callback.

- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  Clears or sets the damage flags.

- void clear_output ()
  Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.

- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

- Fl_Color color () const
  Gets the background color of the widget.

- void color (Fl_Color bg)
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

- Fl_Color color2 () const
  For back compatibility only.

- void color2 (unsigned a)
  For back compatibility only.

- int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.

- void copy_label (const char *new_label)
  Sets the current label.

- void copy_tooltip (const char *text)
  Sets the current tooltip text.

- uchar damage () const
  Returns non-zero if draw() needs to be called.

- void damage (uchar c)
  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- int damage_resize (int, int, int, int)
  Internal use only.

- void deactivate ()
  Deactivates the widget.

- Fl_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

- const Fl_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void **deimage** (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

• void **deimage** (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

• int **deimage_bound** () const
  Returns whether the inactive image is managed by the widget.

• void **do_callback** (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

• void **do_callback** (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void **do_callback** (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

• void **draw_label** (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

• int **h** () const
  Gets the widget height.

• virtual void **hide** ()
  Makes a widget invisible.

• Fl_Image * **image** ()
  Gets the image that is used as part of the widget label when in the active state.

• const Fl_Image * **image** () const
  Gets the image that is used as part of the widget label when in the active state.

• void **image** (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.

• void **image** (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.

• int **image_bound** () const
  Returns whether the image is managed by the widget.

• int **inside** (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.

• int **is_label_copied** () const
  Returns whether the current label was assigned with copy_label().

• const char * **label** () const
  Gets the current label text.

• void **label** (const char *text)
  Sets the current label pointer.

• void **label** (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.

• Fl_Color **labelcolor** () const
  Gets the label color.

• void **labelcolor** (Fl_Color c)
  Sets the label color.

• Fl_Font **labelfont** () const
  Gets the font to use.

• void **labelfont** (Fl_Font f)
  Sets the font to use.

• Fl_Fontsize **labelsize** () const
  Gets the font size in pixels.

• void **labelsize** (Fl_Fontsize pix)
  Sets the font size in pixels.
Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.

• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.

• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.

• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.

• void parent (Fl_Group *p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.

• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.

• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.

• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.

• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.

• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int testShortcut ()
  Returns true if the widget's label contains the entered 'x' shortcut.

• const char * tooltip () const
  Gets the current tooltip text.
• void tooltip (const char *text)
  
  

  Sets the current tooltip text.

• Fl_Window * top_window () const
  
  Returns a pointer to the top-level window for the widget.

• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const
  
  Gets the widget type.

• void type (uchar t)
  
  Sets the widget type.

• int use_accents_menu ()
  
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void * user_data () const
  
  Gets the user data for this widget.

• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  
  Sets the user data for this widget.

• void user_data (void *v)
  
  Sets the user data for this widget.

• unsigned int visible () const
  
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  
  Checks whether this widget has a visible focus.

• void visible_focus (int v)
  
  Modifies keyboard focus navigation.

• int visible_r () const
  
  Returns whether a widget and all its parents are visible.

• int w () const
  
  Gets the widget width.

• Fl_When when () const
  
  Returns the conditions under which the callback is called.

• void when (uchar i)
  
  Sets the flags used to decide when a callback is called.

• Fl_Window * window () const
  
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  
  Gets the widget position in its window.

• int y () const
  
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  
  Destroys the widget.

Static Public Member Functions

• static int kf_backspace (int c, Fl_Text_Editor *e)
  
  Does a backspace for key 'c' in the current buffer of editor 'e'.

• static int kf_c_s_move (int c, Fl_Text_Editor *e)
  
  Extends the current selection in the direction indicated by control key 'c' in editor 'e'.

• static int kf_copy (int c, Fl_Text_Editor *e)
  
  Does a copy of selected text or the current character in the current buffer of editor 'e'.

• static int kf_ctrl_move (int c, Fl_Text_Editor *e)
  
  Moves the current text cursor in the direction indicated by control key 'c' in editor 'e'.
• static int kf_cut (int c, Fl_Text_Editor *e)
  
  Does a cut of selected text in the current buffer of editor 'e'.

• static int kf_default (int c, Fl_Text_Editor *e)
  
  Inserts the text associated with key 'c' in editor 'e'.

• static int kf_delete (int c, Fl_Text_Editor *e)
  
  Does a delete of selected text or the current character in the current buffer of editor 'e'.

• static int kf_down (int c, Fl_Text_Editor *e)
  
  Moves the text cursor one line down for editor 'e'.

• static int kf_end (int c, Fl_Text_Editor *e)
  
  Moves the text cursor to the end of the current line in editor 'e'.

• static int kf_enter (int c, Fl_Text_Editor *e)
  
  Inserts a newline for key 'c' at the current cursor position in editor 'e'.

• static int kf_home (int c, Fl_Text_Editor *e)
  
  Moves the text cursor to the beginning of the current line in editor 'e'.

• static int kf_ignore (int c, Fl_Text_Editor *e)
  
  Ignores the key 'c' in editor 'e'.

• static int kf_insert (int c, Fl_Text_Editor *e)
  
  Toggles the insert mode for editor 'e'.

• static int kf_left (int c, Fl_Text_Editor *e)
  
  Moves the text cursor one character to the left in editor 'e'.

• static int kf_move (int c, Fl_Text_Editor *e)
  
  Moves the text cursor in the direction indicated by key 'c' in editor 'e'.

• static int kf_page_down (int c, Fl_Text_Editor *e)
  
  Moves the text cursor down one page for editor 'e'.

• static int kf_page_up (int c, Fl_Text_Editor *e)
  
  Moves the text cursor up one page for editor 'e'.

• static int kf_paste (int c, Fl_Text_Editor *e)
  
  Does a paste of selected text in the current buffer of editor 'e'.

• static int kf_redo (int c, Fl_Text_Editor *e)
  
  Redo last undo action.

• static int kf_right (int c, Fl_Text_Editor *e)
  
  Moves the text cursor one character to the right for editor 'e'.

• static int kf_select_all (int c, Fl_Text_Editor *e)
  
  Selects all text in the current buffer in editor 'e'.

• static int kf_shift_move (int c, Fl_Text_Editor *e)
  
  Extends the current selection in the direction of key 'c' in editor 'e'.

• static int kf_undo (int c, Fl_Text_Editor *e)
  
  Undoes last edit in the current buffer of editor 'e'.

• static int kf_up (int c, Fl_Text_Editor *e)
  
  Moves the text cursor one line up for editor 'e'.

**Static Public Member Functions inherited from Fl_Group**

• static Fl_Group * current ()
  
  Returns the currently active group.

• static void current (Fl_Group *g)
  
  Sets the current group.
Static Public Member Functions inherited from **Fl_Widget**
- static void **default_callback** (Fl_Widget ∗widget, void ∗data)
  The default callback for all widgets that don’t set a callback.
- static unsigned int **labelShortcut** (const char ∗t)
  Returns the Unicode value of the ‘&x’ shortcut in a given text.
- static int **testShortcut** (const char ∗t, const bool require_alt=false)
  Returns true if the given text ‘t’ contains the entered ‘&x’ shortcut.

Protected Member Functions
- int **handleKey** ()
  Handles a key press in the editor.
- void **maybeDoCallback** (Fl_Callback_Reason reason=FL_REASON_CHANGED)
  does or does not a callback according to changed() and when() settings

Protected Member Functions inherited from **Fl_Text_Display**
- void **absoluteTopLineNumber** (int oldFirstChar)
  Re-calculate absolute top line number for a change in scroll position.
- void **calcLastChar** ()
  Update last display character index.
- void **calcLineStarts** (int startLine, int endLine)
  Update the line starts array.
- void **clearRect** (int style, int x, int y, int width, int height) const
  Clear a rectangle with the appropriate background color for style.
- void **displayInsert** ()
  Scroll the display to bring insertion cursor into view.
- void **draw** () FL_OVERRIDE
  Draw the widget.
- void **drawCursor** (int, int)
  Draw a cursor with top center at X, Y.
- void **drawLineNumbers** (bool clearAll)
  Refresh the line number area.
- void **drawRange** (int start, int end)
  Draw a range of text.
- void **drawString** (int style, int x, int y, int toX, const char ∗string, int nChars) const
  Draw a text segment in a single style.
- void **drawText** (int X, int Y, int W, int H)
  Refresh a rectangle of the text display.
- void **drawVLine** (int visLineNum, int leftClip, int rightClip, int leftCharIndex, int rightCharIndex)
  Draw a single line of text.
- int **emptyVlines** () const
  Return true if there are lines visible with no corresponding buffer text.
- void **extendRangeForStyles** (int ∗start, int ∗end)
  I don’t know what this does!
- void **findLineEnd** (int pos, bool start_pos_is_line_start, int ∗lineEnd, int ∗nextLineStart) const
  Finds both the end of the current line and the start of the next line.
- void **findWrapRange** (const char ∗deletedText, int pos, int nInserted, int nDeleted, int ∗modRangeStart, int ∗modRangeEnd, int ∗linesInserted, int ∗linesDeleted)
  Wrapping calculations.
- int **findX** (const char ∗s, int len, int style, int x) const
Find the index of the character that lies at the given x position / closest cursor position.

- **int get_absolute_top_line_number () const**
  Returns the absolute (non-wrapped) line number of the first line displayed.

- **int handle_rmb (int readonly)**
  Handle right mouse button down events.

- **int handle_vline (int mode, int lineStart, int lineLen, int leftChar, int rightChar, int topClip, int bottomClip, int leftClip, int rightClip) const**
  Universal pixel machine.

- **int longest_vline () const**
  Find the longest line of all visible lines.

- **void maintain_absolute_top_line_number (int state)**
  Line numbering stuff, currently unused.

- **int maintaining_absolute_top_line_number () const**
  Returns true if a separate absolute top line number is being maintained.

- **void measure_deleted_lines (int pos, int nDeleted)**
  Wrapping calculations.

- **double measure_proportional_character (const char ∗s, int colNum, int pos) const**
  Wrapping calculations.

- **int measure_vline (int visLineNum) const**
  Returns the width in pixels of the displayed line pointed to by "visLineNum".

- **void offset_line_starts (int newTopLineNum)**
  Offset line start counters for a new vertical scroll position.

- **int position_to_line (int pos, int ∗lineNum) const**
  Convert a position index into a line number offset.

- **int position_to_linecol (int pos, int ∗lineNum, int ∗column) const**
  Find the line and column number of position pos.

- **void reset_absolute_top_line_number ()**
  Reestablish the absolute (non-wrapped) top line number.

- **int scroll_ (int topLineNum, int horizOffset)**
  Scrolls the current buffer to start at the specified line and column.

- **double string_width (const char ∗string, int length, int style) const**
  Find the width of a string in the font of a particular style.

- **void update_h_scrollbar ()**
  Update horizontal scrollbar.

- **void update_line_starts (int pos, int charInserted, int charDeleted, int linesInserted, int linesDeleted, int ∗scrolled)**
  Update line start arrays and variables.

- **void update_v_scrollbar ()**
  Update vertical scrollbar.

- **int vline_length (int visLineNum) const**
  Count number of bytes in a visible line.

- **int wrapUsesCharacter (int lineEndPos) const**
  Check if the line break is caused by a newline or by line wrapping.

- **void wrapped_line_counter (FL_Text_Buffer ∗buf, int startPos, int maxPos, int maxLines, bool startPosLineStart, int styleBufOffset, int ∗retPos, int ∗retLines, int ∗retLineStart, int ∗retLineEnd, bool countLastLineMissingNewLine=true) const**
  Wrapping calculations.

- **int xy_to_position (int x, int y, int PosType=CHARACTER_POS) const**
  Translate a pixel position into a character index.

- **void xy_to_rowcol (int x, int y, int ∗row, int ∗column, int PosType=CHARACTER_POS) const**
  Translate pixel coordinates into row and column.
Protected Member Functions inherited from Fl_Group

- \texttt{Fl\_Rect\ *} \texttt{bounds()} \newline
  Returns the internal array of widget sizes and positions.
- void \texttt{draw\_child(Fl\_Widget} \&\texttt{widget}) \texttt{const} \newline
  Forces a child to redraw.
- void \texttt{draw\_children()} \newline
  Draws all children of the group.
- void \texttt{draw\_outside\_label}(const Fl\_Widget \&\texttt{widget}) \texttt{const} \newline
  Parents normally call this to draw outside labels of child widgets.
- virtual int \texttt{on\_insert(Fl\_Widget} \&\texttt{w}, \texttt{int} \texttt{)} \newline
  Allow derived groups to act when a widget is added as a child.
- virtual int \texttt{on\_move}(\texttt{int}, \texttt{int}) \newline
  Allow derived groups to act when a widget is moved within the group.
- virtual void \texttt{on\_remove}(\texttt{int}) \newline
  Allow derived groups to act when a child widget is removed from the group.
- int \* \texttt{sizes()} \newline
  Returns the internal array of widget sizes and positions.
- void \texttt{update\_child(Fl\_Widget} \&\texttt{widget}) \texttt{const} \newline
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void \texttt{clear\_flag}(\texttt{unsigned int} \texttt{c}) \newline
  Clears a flag in the flags mask.
- void \texttt{draw\_backdrop()} \texttt{const} \newline
  If \texttt{FL\_ALIGN\_IMAGE\_BACKDROP} is set, the image or deimage will be drawn.
- void \texttt{draw\_box()} \texttt{const} \newline
  Draws the widget box according its box style.
- void \texttt{draw\_box(Fl\_Boxtype} \texttt{t}, Fl\_Color \texttt{c}) \texttt{const} \newline
  Draws a box of type \texttt{t}, of color \texttt{c} at the widget's position and size.
- void \texttt{draw\_box(Fl\_Boxtype} \texttt{t}, \texttt{int} \texttt{x}, \texttt{int} \texttt{y}, \texttt{int} \texttt{w}, \texttt{int} \texttt{h, Fl\_Color} \texttt{c}) \texttt{const} \newline
  Draws a box of type \texttt{t}, of color \texttt{c} at the position \texttt{X,Y} and size \texttt{W,H}.
- void \texttt{draw\_focus()} \texttt{const} \newline
  Draws a focus rectangle around the widget.
- void \texttt{draw\_focus(Fl\_Boxtype} \texttt{t}, \texttt{int} \texttt{X}, \texttt{int} \texttt{Y}, \texttt{int} \texttt{W}, \texttt{int} \texttt{H}) \texttt{const} \newline
  Draws a focus rectangle around the widget.
- void \texttt{draw\_focus(Fl\_Boxtype} \texttt{t}, \texttt{int} \texttt{x}, \texttt{int} \texttt{y}, \texttt{int} \texttt{w, int} \texttt{h, Fl\_Color} \texttt{bg}) \texttt{const} \newline
  Draws a focus box for the widget at the given position and size.
- void \texttt{draw\_label()} \texttt{const} \newline
  Draws the widget's label at the defined label position.
- void \texttt{draw\_label}(\texttt{int}, \texttt{int}, \texttt{int}, \texttt{int}) \texttt{const} \newline
  Draws the label in an arbitrary bounding box.
- Fl\_Widget(\texttt{int} \texttt{x, int} \texttt{y, int} \texttt{w, int} \texttt{h, const} char \texttt{*label}=0L) \newline
  Creates a widget at the given position and size.
- unsigned int \texttt{flags()} \texttt{const} \newline
  Gets the widget flags mask.
- void \texttt{h(}int \texttt{v)} \newline
  Internal use only.
- void \texttt{set\_flag}(\texttt{unsigned int} \texttt{c}) \newline
  Sets a flag in the flags mask.
- void \texttt{w(}int \texttt{v)}
• void x (int v)  
  *Internal use only.*
• void y (int v)  
  *Internal use only.*

**Static Protected Attributes**

• static Key_Binding *global_key_bindings  
  *Global key binding list.*

**Additional Inherited Members**

**Protected Types inherited from Fl_Text_Display**

• enum {
  DRAW_LINE, FIND_INDEX, FIND_INDEX_FROM_ZERO, GET_WIDTH,  
  FIND_CURSOR_INDEX
}

**Protected Types inherited from Fl_Widget**

• enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,  
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,  
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,  
  MENU_WINDOW = 1<<12, TOOLTIPT_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,  
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,  
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,  
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

**Static Protected Member Functions inherited from Fl_Text_Display**

• static void buffer_modified_cb (int pos, int nInserted, int nDeleted, int nRestyled, const char *deletedText,  
  void *cbArg)  
  *This is called whenever the buffer is modified.*
• static void buffer_predelete_cb (int pos, int nDeleted, void *cbArg)  
  *This is called before any characters are deleted.*
• static void h_scrollbar_cb (Fl_Scrollbar *w, Fl_Text_Display *d)  
  *Callback for drag or valueChanged on horizontal scrollbar.*
• static void scroll_timer_cb (void *)  
  *Timer callback for scroll events.*
• static void v_scrollbar_cb (Fl_Scrollbar *w, Fl_Text_Display *d)  
  *Callback for drag or valueChanged on vertical scrollbar.*

**Protected Attributes inherited from Fl_Text_Display**

• int damage_range1_end  
• int damage_range1_start  
• int damage_range2_end  
• int damage_range2_start
• int display_insert_position_hint
• int dragging
• int dragPos
• int dragType
  • Fl_Color grammar_underline_color_
  • Fl_Align linenumber_align_
  • Fl_Color linenumberbgcolor_
  • Fl_Color linenumberbgcolor_
  • Fl_Font linenumberfont_
  • const char * linenumberformat_
  • Fl_Fontsize linenumbersize_
• int mAbsTopLineNum
  • Fl_Text_Buffer * mBuffer
• double mColumnScale
• int mContinuousWrap
  • Fl_Color mCursor_color
• int mCursorOldY
• int mCursorOn
• int mCursorPos
• int mCursorPreferredXPos
• int mCursorStyle
• int mCursorToHint
• int mFirstChar
• void * mHighlightCBArg
• int mHorizOffset
• int mHorizOffsetHint
  • Fl_Scrollbar * mHScrollbar
• int mLastChar
• int mLineNumLeft
• int mLineNumWidth
• int * mLineStarts
• int mMaxsize
• int mModifyingTabDistance
• int mNBufferLines
• int mNeedAbsTopLineNum
• int mNLinesDeleted
• int mNStyles
• int mNVisibleLines
  • Fl_Text_Buffer * mStyleBuffer
• const Style_Table_Entry * mStyleTable
• int mSuppressResync
• int mTopLineNum
• int mTopLineNumHint
• Unfinished_Style_Cb mUnfinishedHighlightCB
• char mUnfinishedStyle
  • Fl_Scrollbar * mVScrollBar
• int mWrapMarginPix
  • Fl_Align scrollbar_align_
• int scrollbar_width_
  • Fl_Color secondary_selection_color_
• int shortcut_
  • Fl_Color spelling_underline_color_
• struct {
  int h
  int w
  int x
  int y
} text_area

• Fl_Color textcolor_
• Fl_Font textfont_
• Fl_Fontsize textsize_

12.144.1 Detailed Description
This is the FLTK text editor widget.
It allows the user to edit multiple lines of text and supports highlighting and scrolling. The buffer that is displayed in
the widget is managed by the Fl_Text_Buffer class.

12.144.2 Member Function Documentation
12.144.2.1 add_key_binding()

void Fl_Text_Editor::add_key_binding (  
  int key,
  int state,
  Key_Func function,
  Key_Binding ** list )

Adds a key of state state with the function function to an arbitrary key binding list list.
This can be used in derived classes to add global key bindings by using the global (static) Key_Binding list
Fl_Text_Editor::global_key_bindings.

12.144.2.2 handle()

int Fl_Text_Editor::handle (  
  int e ) [virtual]

Event handling.
Reimplemented from Fl_Text_Display.

12.144.2.3 insert_mode() [1/2]

int Fl_Text_Editor::insert_mode ( ) [inline]

Gets the current insert mode; if non-zero, new text is inserted before the current cursor position.
Otherwise, new text replaces text at the current cursor position.

12.144.2.4 insert_mode() [2/2]

void Fl_Text_Editor::insert_mode (  
  int b ) [inline]

Sets the current insert mode; if non-zero, new text is inserted before the current cursor position.
Otherwise, new text replaces text at the current cursor position.

12.144.2.5 kf_backspace()

int Fl_Text_Editor::kf_backspace (  
  int c,
  Fl_Text_Editor * e ) [static]

Does a backspace for key 'c' in the current buffer of editor 'e'.
Any current selection is deleted. Otherwise, the character left is deleted and the cursor moved. The key value 'c'
is currently unused.
12.144.2.6 kf_c_s_move()

int Fl_Text_Editor::kf_c_s_move (  
    int c,  
    Fl_Text_Editor * e ) [static]
Extends the current selection in the direction indicated by control key 'c' in editor 'e'.
See also
    kf_ctrl_move().

12.144.2.7 kf_copy()

int Fl_Text_Editor::kf_copy (  
    int c,  
    Fl_Text_Editor * e ) [static]
Does a copy of selected text or the current character in the current buffer of editor 'e'.
The key value 'c' is currently unused.

12.144.2.8 kf_ctrl_move()

int Fl_Text_Editor::kf_ctrl_move (  
    int c,  
    Fl_Text_Editor * e ) [static]
Moves the current text cursor in the direction indicated by control key 'c' in editor 'e'.
Supported values for 'c' are currently:
    FL_Home -- moves the cursor to the beginning of the document
    FL_End -- moves the cursor to the end of the document
    FL_Left -- moves the cursor left one word
    FL_Right -- moves the cursor right one word
    FL_Up -- scrolls up one line, without moving cursor
    FL_Down -- scrolls down one line, without moving cursor
    FL_Page_Up -- moves the cursor to the beginning of the top line on the current page
    FL_Page_Down -- moves the cursor to the beginning of the last line on the current page

12.144.2.9 kf_cut()

int Fl_Text_Editor::kf_cut (  
    int c,  
    Fl_Text_Editor * e ) [static]
Does a cut of selected text in the current buffer of editor 'e'.
The key value 'c' is currently unused.

12.144.2.10 kf_default()

int Fl_Text_Editor::kf_default (  
    int c,  
    Fl_Text_Editor * e ) [static]
Inserts the text associated with key 'c' in editor 'e'.
Honors the current selection and insert/overstrike mode.

12.144.2.11 kf_delete()

int Fl_Text_Editor::kf_delete (  
    int c,  
    Fl_Text_Editor * e ) [static]
Does a delete of selected text or the current character in the current buffer of editor 'e'.
The key value 'c' is currently unused.
12.144.2.12  kf_down()

int Fl_Text_Editor::kf_down (  
    int c,  
    Fl_Text_Editor * e ) [static]

Moves the text cursor one line down for editor 'e'.
Same as kf_move(FL_Down, e). The key value 'c' is currently unused.

12.144.2.13  kf_end()

int Fl_Text_Editor::kf_end (  
    int c,  
    Fl_Text_Editor * e ) [static]

Moves the text cursor to the end of the current line in editor 'e'.
Same as kf_move(FL_End, e). The key value 'c' is currently unused.

12.144.2.14  kf_enter()

int Fl_Text_Editor::kf_enter (  
    int c,  
    Fl_Text_Editor * e ) [static]

Inserts a newline for key 'c' at the current cursor position in editor 'e'.
The key value 'c' is currently unused.

12.144.2.15  kf_home()

int Fl_Text_Editor::kf_home (  
    int c,  
    Fl_Text_Editor * e ) [static]

Moves the text cursor to the beginning of the current line in editor 'e'.
Same as kf_move(FL_Home, e). The key value 'c' is currently unused.

12.144.2.16  kf_ignore()

int Fl_Text_Editor::kf_ignore (  
    int c,  
    Fl_Text_Editor * e ) [static]

Ignores the key 'c' in editor 'e'.
This method can be used as a keyboard binding to disable a key that might otherwise be handled or entered as text.
An example would be disabling FL_Escape, so that it isn't added to the buffer when invoked by the user.

12.144.2.17  kf_insert()

int Fl_Text_Editor::kf_insert (  
    int c,  
    Fl_Text_Editor * e ) [static]

Toggles the insert mode for editor 'e'.
The key value 'c' is currently unused.

12.144.2.18  kf_left()

int Fl_Text_Editor::kf_left (  
    int c,  
    Fl_Text_Editor * e ) [static]

Moves the text cursor one character to the left in editor 'e'.
Same as kf_move(FL_Left, e). The key value 'c' is currently unused.
12.144.2.19  kf_m_s_move()

int Fl_Text_Editor::kf_m_s_move (    
    int c,    
    Fl_Text_Editor * e ) [static]

Extends the current selection in the direction indicated by meta key 'c' in editor 'e'.

See also

kf_meta_move().

12.144.2.20  kf_meta_move()

int Fl_Text_Editor::kf_meta_move (    
    int c,    
    Fl_Text_Editor * e ) [static]

Moves the current text cursor in the direction indicated by meta key 'c' in editor 'e'.

Supported values for 'c' are currently:

- FL_Up -- moves cursor to the beginning of the current document
- FL_Down -- moves cursor to the end of the current document
- FL_Left -- moves the cursor to the beginning of the current line
- FL_Right -- moves the cursor to the end of the current line

12.144.2.21  kf_move()

int Fl_Text_Editor::kf_move (    
    int c,    
    Fl_Text_Editor * e ) [static]

Moves the text cursor in the direction indicated by key 'c' in editor 'e'.

Supported values for 'c' are currently:

- FL_Home -- moves the cursor to the beginning of the current line
- FL_End -- moves the cursor to the end of the current line
- FL_Left -- moves the cursor left one character
- FL_Right -- moves the cursor right one character
- FL_Up -- moves the cursor up one line
- FL_Down -- moves the cursor down one line
- FL_Page_Up -- moves the cursor up one page
- FL_Page_Down -- moves the cursor down one page

12.144.2.22  kf_page_down()

int Fl_Text_Editor::kf_page_down (    
    int c,    
    Fl_Text_Editor * e ) [static]

Moves the text cursor down one page for editor 'e'.

Same as kf_move(FL_Page_Down, e). The key value 'c' is currently unused.

12.144.2.23  kf_page_up()

int Fl_Text_Editor::kf_page_up (    
    int c,    
    Fl_Text_Editor * e ) [static]

Moves the text cursor up one page for editor 'e'.

Same as kf_move(FL_Page_Up, e). The key value 'c' is currently unused.

12.144.2.24  kf_paste()

int Fl_Text_Editor::kf_paste (    
    int c,    
    Fl_Text_Editor * e ) [static]

Does a paste of selected text in the current buffer of editor 'e'.

Any current selection is replaced with the pasted content. The key value 'c' is currently unused.
12.144.2.25  kf_redo()

int Fl_Text_Editor::kf_redo (  
    int c,  
    Fl_Text_Editor * e ) [static]

Redo last undo action.  
Also deselects previous selection. The key value 'c' is currently unused.

12.144.2.26  kf_right()

int Fl_Text_Editor::kf_right (  
    int c,  
    Fl_Text_Editor * e ) [static]

Moves the text cursor one character to the right for editor 'e'.  
Same as kf_move(FL_Right, e). The key value 'c' is currently unused.

12.144.2.27  kf_select_all()

int Fl_Text_Editor::kf_select_all (  
    int c,  
    Fl_Text_Editor * e ) [static]

Selects all text in the current buffer in editor 'e'.  
The key value 'c' is currently unused.

12.144.2.28  kf_shift_move()

int Fl_Text_Editor::kf_shift_move (  
    int c,  
    Fl_Text_Editor * e ) [static]

Extends the current selection in the direction of key 'c' in editor 'e'.  
See also  
    kf_move()

12.144.2.29  kf_undo()

int Fl_Text_Editor::kf_undo (  
    int c,  
    Fl_Text_Editor * e ) [static]

Undo last edit in the current buffer of editor 'e'.  
Also deselects previous selection. The key value 'c' is currently unused.

12.144.2.30  kf_up()

int Fl_Text_Editor::kf_up (  
    int c,  
    Fl_Text_Editor * e ) [static]

Moves the text cursor one line up for editor 'e'.  
Same as kf_move(FL_Up, e). The key value 'c' is currently unused.

12.144.2.31  remove_key_binding()

void Fl_Text_Editor::remove_key_binding {  
    int key,  
    int state,  
    Key_Binding ** list )

Removes the key binding associated with the key key of state state from the Key_Binding list list.  
This can be used in derived classes to remove global key bindings by using the global (static) Key_Binding list Fl_Text_Editor::global_keyBindings.
12.144.2.32  tab_nav() [1/2]

int Fl_Text_Editor::tab_nav ( ) const
Check if Tab focus navigation is enabled.
If disabled (default), hitting Tab inserts a tab character into the editor buffer.
If enabled, hitting Tab navigates focus to the next widget, and Shift-Tab navigates focus to the previous widget.

Returns
if Tab inserts tab characters or moves the focus

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>Tab inserts tab characters (default)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tab navigation is enabled.</td>
</tr>
</tbody>
</table>

See also
tab_nav(int), Fl::OPTION_ARROW_FOCUS.

Version
1.3.4 ABI feature

12.144.2.33  tab_nav() [2/2]

void Fl_Text_Editor::tab_nav ( int val )
Enables or disables Tab key focus navigation.
When disabled (default), tab characters are inserted into Fl_Text_Editor. Only the mouse can change focus. This behavior is desirable when Fl_Text_Editor is used, e.g. in a source code editor.
When enabled, Tab navigates focus to the next widget, and Shift-Tab navigates focus to the previous widget. This behavior is desirable when Fl_Text_Editor is used e.g. in a database input form.
Currently, this method is implemented as a convenience method that adjusts the key bindings for the Tab key. This implementation detail may change in the future. Know that changing the editor's key bindings for Tab and Shift-Tab may affect tab navigation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
<th>If val is 0, Tab inserts a tab character (default).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>If val is 1, Tab navigates widget focus.</td>
</tr>
</tbody>
</table>

See also
tab_nav(), Fl::OPTION_ARROW_FOCUS.

Version
1.3.4 ABI feature

12.144.3  Member Data Documentation

12.144.3.1  global_key_bindings

Key_Binding* Fl_Text_Editor::global_key_bindings [static], [protected]
Global key binding list.
Derived classes can add key bindings for all Fl_Text_Editor widgets by adding a Key_Binding to this list.
See also

```cpp
add_key_binding(int key, int state, Key_Func f, Key_Binding** list);
```

The documentation for this class was generated from the following files:

- Fl_Text_Editor.H
- Fl_Text_Editor.hxx

**12.145 Fl_Text_Selection Class Reference**

This is an internal class for Fl_Text_Buffer to manage text selections.
```cpp
#include <Fl_Text_Buffer.H>
```

**Public Member Functions**

- **int end() const**
  Returns the byte offset to the character after the last selected character.
- **int includes (int pos) const**
  Returns true if position `pos` is in the Fl_Text_Selection.
- **int length() const**
  Returns the size in bytes of the selection.
- **int position (int *startpos, int *endpos) const**
- **bool selected() const**
  Returns true if any text is selected.
- **void selected (bool b)**
  Modifies the 'selected' flag.
- **int selected (int *startpos, int *endpos) const**
  Returns the status and the positions of this selection.
- **void set (int startpos, int endpos)**
  Sets the selection range.
- **int start() const**
  Returns the byte offset to the first selected character.
- **void update (int pos, int nDeleted, int nInserted)**
  Updates a selection after text was modified.

**Protected Attributes**

- **int mEnd**
  byte offset to the character after the last selected character
- **bool mSelected**
  this flag is set if any text is selected
- **int mStart**
  byte offset to the first selected character

**Friends**

- **class Fl_Text_Buffer**
12.145.1 Detailed Description

This is an internal class for Fl_Text_Buffer to manage text selections. All methods use byte (not UTF-8 character) offsets and start at 0. This class works correctly with UTF-8 strings assuming that the parameters for all calls are on character boundaries.

If the selection is inactive (not currently used), then selected() returns false and start() and end() return 0 (zero). The stored offsets are in ascending order, hence the following conditions are true (pseudo code):

```cpp
if (!selected()) : (start() == 0) && (end() == 0) && (start() == end())
if (selected()) : start() < end()
always : 0 <= start() <= end()
always : length() == end() - start()
```

The selection size in bytes can always (unconditionally) be computed by

```cpp
int size = sel->end() - sel->start();
```

See also

`length()`

Note

The protected member variables mStart and mEnd are not necessarily 0 (zero) if mSelected == false because they are not cleared when selected(false) is called (as of Jul 2017). This may be changed in the future.

12.145.2 Member Function Documentation

12.145.2.1 end()

```cpp
int Fl_Text_Selection::end ( ) const [inline]
```

Returns the byte offset to the character after the last selected character. The returned offset is only valid if selected() returns true (non-zero). The offset is 0 if no text is selected (since FLTK 1.4.0).

Note

In FLTK 1.3.x the returned offset could be non-zero even if selected() would have returned 0.

Returns

byte offset or 0 if not selected.

12.145.2.2 includes()

```cpp
int Fl_Text_Selection::includes ( int pos ) const
```

Returns true if position pos is in the Fl_Text_Selection. pos must be at a character boundary.

12.145.2.3 length()

```cpp
int Fl_Text_Selection::length ( ) const [inline]
```

Returns the size in bytes of the selection. This is a convenience method. It always returns the same as end() - start() and it returns 0 if selected() == false.

Returns

size in bytes or 0 if not selected.

Since

FLTK 1.4.0

---

Generated by Doxygen
12.145.2.4  position()

```cpp
int Fl_Text_Selection::position (  
    int * startpos,  
    int * endpos ) const [inline]
```

**Deprecated** "in 1.4.0 - use selected(startpos, endpos) instead"

12.145.2.5  selected() [1/3]

```cpp
bool Fl_Text_Selection::selected ( ) const [inline]
```

Returns true if any text is selected.

Returns

true if any text has been selected, or false if no text is selected.

12.145.2.6  selected() [2/3]

```cpp
void Fl_Text_Selection::selected (  
    bool b ) [inline]
```

Modifies the 'selected' flag.

Parameters

<table>
<thead>
<tr>
<th>b</th>
<th>new flag</th>
</tr>
</thead>
</table>

12.145.2.7  selected() [3/3]

```cpp
int Fl_Text_Selection::selected (  
    int * startpos,  
    int * endpos ) const
```

Returns the status and the positions of this selection.

This method returns the same as selected() as an int (0 or 1) in its return value and the offsets to the start of the selection in startpos and to the byte after the last selected character in endpos, if selected() is true. If selected() is false, both offsets are set to 0.

**Note**

In FLTK 1.3.x startpos and endpos were not modified if selected() was false.

Parameters

<table>
<thead>
<tr>
<th>startpos</th>
<th>return byte offset to first selected character</th>
</tr>
</thead>
<tbody>
<tr>
<td>endpos</td>
<td>return byte offset pointing after last selected character</td>
</tr>
</tbody>
</table>

Returns

whether the selection is active (selected()) or not

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>if not selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if selected</td>
</tr>
</tbody>
</table>
See also

- selected(), start(), end()

### 12.145.2.8 set()

```cpp
void Fl_Text_Selection::set ( int startpos, int endpos )
```

Sets the selection range. startpos and endpos must be at a character boundary. If startpos \(!=\) endpos selected() is set to true, else to false. If startpos is greater than endpos they are swapped so that startpos \(\leq\) endpos.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>startpos</th>
<th>byte offset to first selected character</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>endpos</td>
<td>byte offset pointing after last selected character</td>
</tr>
</tbody>
</table>

### 12.145.2.9 start()

```cpp
int Fl_Text_Selection::start ( ) const [inline]
```

Returns the byte offset to the first selected character. The returned offset is only valid if selected() returns true. If the selection is not valid the returned offset is 0 since FLTK 1.4.0.

**Note**

In FLTK 1.3.x the returned offset could be non-zero even if selected() would have returned 0.

**Returns**

byte offset or 0 if not selected.

### 12.145.2.10 update()

```cpp
void Fl_Text_Selection::update ( int pos, int nDeleted, int nInserted )
```

Updates a selection after text was modified. Updates an individual selection for changes in the corresponding text.

**Parameters**

<table>
<thead>
<tr>
<th>pos</th>
<th>byte offset into text buffer at which the change occurred</th>
</tr>
</thead>
<tbody>
<tr>
<td>nDeleted</td>
<td>number of bytes deleted from the buffer</td>
</tr>
<tr>
<td>nInserted</td>
<td>number of bytes inserted into the buffer</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Text_Buffer.H
- Fl_Text_Buffer.cxx

## 12.146 Fl_Tile Class Reference

The `Fl_Tile` class lets you resize its children by dragging the border between them.
Inheritance diagram for Fl_Tile:

```
Fl_Widget
   ↓
Fl_Group
   ↓
Fl_Tile
```

**Classes**

- struct `Size_Range`

**Public Member Functions**

- virtual void `drag_intersection` (int `oldx`, int `oldy`, int `newx`, int `newy`)
  
  Drags the intersection at `(oldx, oldy)` to `(newx, newy)`.

- `Fl_Tile` (int `X`, int `Y`, int `W`, int `H`, const char ∗`L`=0)

  Creates a new `Fl_Tile` widget using the given position, size, and label string.

- int `handle` (int `event`) FL_OVERRIDE

  Handles the specified event.

- void `init_size_range` (int `default_min_w`=-1, int `default_min_h`=-1)

  Initialize the size range mode of `Fl_Tile` and set the default minimum width and height.

- virtual void `move_intersection` (int `oldx`, int `oldy`, int `newx`, int `newy`)

  Drags the intersection at `(oldx, oldy)` to `(newx, newy)`.

- void `position` (int `oldx`, int `oldy`, int `newx`, int `newy`)

- void `position` (int `x`, int `y`)

- void `resize` (int `X`, int `Y`, int `W`, int `H`) FL_OVERRIDE

  Resizes the `Fl_Tile` widget and its children.

- void `size_range` (Fl_Widget ∗`w`, int `minw`, int `minh`, int `maxw`=0x7FFFFFFF, int `maxh`=0x7FFFFFFF)

  Set the allowed size range for the give child widget.

- void `size_range` (int `index`, int `minw`, int `minh`, int `maxw`=0x7FFFFFFF, int `maxh`=0x7FFFFFFF)

  Set the allowed size range for the child at the given index.

- ∼`Fl_Tile` () FL_OVERRIDE

  Destructor.

**Public Member Functions inherited from Fl_Group**

- Fl_Widget ∗& _ddfdesign_kludge ()

  This is for forms compatibility only.

- void `add` (Fl_Widget ∗)

  The widget is removed from its current group (if any) and then added to the end of this group.

- void `add` (Fl_Widget ∗`o`)

  See void Fl_Group::add(Fl_Widget ∗`w`)

- void `add_resizable` (Fl_Widget ∗`w`)

  Adds a widget to the group and makes it the resizable widget.

- Fl_Widget ∗`const` ∗`array` () const

  Returns a pointer to the array of children.

- Fl_Group ∗`const` ∗`as_group` () const FL_OVERRIDE

- Fl_Group ∗`as_group` () FL_OVERRIDE

  Returns an Fl_Group pointer if this widget is an Fl_Group.

- void `begin` ()
Sets the current group so you can build the widget tree by just constructing the widgets.

- `Fl_Widget * child (int n) const`
  Returns `array()[n].`

- `int children () const`
  Returns how many child widgets the group has.

- `void clear ()`
  Deletes all child widgets from memory recursively.

- `unsigned int clip_children ()`
  Returns the current clipping mode.

- `void clip_children (int c)`
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- `virtual int delete_child (int n)`
  Removes the widget at index from the group and deletes it.

- `void end ()`
  Exactly the same as current(this->parent()).

- `int find (const Fl_Widget &o) const`
  See int Fl_Group::find(const Fl_Widget *w) const.

- `int find (const Fl_Widget *) const`
  Searches the child array for the widget and returns the index.

- `Fl_Group (int, int, int, int, const char *)=0)`
  Creates a new Fl_Group widget using the given position, size, and label string.

- `void focus (Fl_Widget *)`

- `void forms_end ()`
  This is for forms compatibility only.

- `void init_sizes ()`
  Resets the internal array of widget sizes and positions.

- `void insert (Fl_Widget &, int i)`
  The widget is removed from its current group (if any) and then inserted into this group.

- `void insert (Fl_Widget &o, Fl_Widget *) before)`
  This does insert(w, find(before)).

- `void remove (Fl_Widget &)`
  Removes a widget from the group but does not delete it.

- `void remove (Fl_Widget *) o)`
  Removes the widget o from the group.

- `void remove (int index)`
  Removes the widget at index from the group but does not delete it.

- `Fl_Widget * resizable () const`
  Returns the group's resizable widget.

- `void resizable (Fl_Widget &o)`
  Sets the group's resizable widget.

- `void resizable (Fl_Widget *) o)`
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- `virtual ~Fl_Group ()`
  The destructor also deletes all the children.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  
  Activates the widget.
- unsigned int active () const
  
  Returns whether the widget is active.
- int active_r () const
  
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  
  Gets the label alignment.
- void align (Fl_Align alignment)
  
  Sets the label alignment.
- long argument () const
  
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ as_gl_window () const
- virtual Fl_Window ∗ as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ as_window () const
- void bind_deimage (Fl_Image ∗ img)
  
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗ img)
  
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  
  Sets the box type for the widget.
- Fl_Callback ∗ p callback () const
  
  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb)
  
  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗ cb, void ∗ p)
  
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗ cb)
  
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗ cb, long p=0)
  
  Sets the current callback function for the widget.
- unsigned int changed () const
  
  Checks if the widget value changed since the last callback.
- void clear_active ()
Marks the widget as inactive without sending events or changing focus.
  • void clear_changed ()
    Marks the value of the widget as unchanged.
  • void clear_damage (uchar c=0)
    Clears or sets the damage flags.
  • void clear_output ()
    Sets a widget to accept input.
  • void clear_visible ()
    Hides the widget.
  • void clear_visible_focus ()
    Disables keyboard focus navigation with this widget.
  • Fl_Color color () const
    Gets the background color of the widget.
  • void color (Fl_Color bg)
    Sets the background color of the widget.
  • void color (Fl_Color bg, Fl_Color sel)
    Sets the background and selection color of the widget.
  • Fl_Color color2 () const
    For back compatibility only.
  • void color2 (unsigned a)
    For back compatibility only.
  • int contains (const Fl_Widget ∗w) const
    Checks if w is a child of this widget.
  • void copy_label (const char ∗new_label)
    Sets the current label.
  • void copy_tooltip (const char ∗text)
    Sets the current tooltip text.
  • uchar damage () const
    Returns non-zero if draw() needs to be called.
  • void damage (uchar c)
    Sets the damage bits for the widget.
  • void damage (uchar c, int x, int y, int w, int h)
    Sets the damage bits for an area inside the widget.
  • int damage_resize (int, int, int, int)
    Internal use only.
  • void deactivate ()
    Deactivates the widget.
  • Fl_Image ∗deimage ()
    Gets the image that is used as part of the widget label when in the inactive state.
  • const Fl_Image ∗deimage () const
    Gets the image that is used as part of the widget label when in the inactive state.
  • void deimage (Fl_Image &img)
    Sets the image to use as part of the widget label when in the inactive state.
  • void deimage (Fl_Image ∗img)
    Sets the image to use as part of the widget label when in the inactive state.
  • int deimage_bound () const
    Returns whether the inactive image is managed by the widget.
  • void do_callback (FL_Callback_Reason reason=FL_REASON_UNKNOWN)
    Calls the widget callback function with default arguments.
  • void do_callback (Fl_Widget ∗w, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
    Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget ∗wgt, void ∗arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image ∗image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image ∗image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget ∗wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char ∗label () const
  Gets the current label text.
• void label (const char ∗text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char ∗b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
Returns if a widget is used for output only.

- `Fl_Group * parent () const`
  Returns a pointer to the parent widget.

- `void parent (Fl_Group * p)`
  Internal use only - "for hacks only".

- `void position (int X, int Y)`
  Repositions the window or widget.

- `void redraw ()`
  Schedules the drawing of the widget.

- `void redraw_label ()`
  Schedules the drawing of the label.

- `Fl_Color selection_color () const`
  Gets the selection color.

- `void selection_color (Fl_Color a)`
  Sets the selection color.

- `void set_active ()`
  Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  Marks the value of the widget as changed.

- `void set_output ()`
  Sets a widget to output only.

- `void set_visible ()`
  Makes the widget visible.

- `void set_visible_focus ()`
  Enables keyboard focus navigation with this widget.

- `int shortcut_label () const`
  Returns whether the widget's label uses '&' to indicate shortcuts.

- `void shortcut_label (int value)`
  Sets whether the widget's label uses '&' to indicate shortcuts.

- `virtual void show ()`
  Makes a widget visible.

- `void size (int W, int H)`
  Changes the size of the widget.

- `int take_focus ()`
  Gives the widget the keyboard focus.

- `unsigned int takesevents () const`
  Returns if the widget is able to take events.

- `int test_shortcut ()`
  Returns true if the widget's label contains the entered 'x' shortcut.

- `const char * tooltip () const`
  Gets the current tooltip text.

- `void tooltip (const char * text)`
  Sets the current tooltip text.

- `Fl_Window * top_window () const`
  Returns a pointer to the top-level window for the widget.

- `Fl_Window * top_window_offset (int &xoff, int &yoff) const`
  Finds the x/y offset of the current widget relative to the top-level window.

- `uchar type () const`
  Gets the widget type.

- `void type (uchar t)`
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void * user_data () const
  Gets the user data for this widget.

• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

• void user_data (void *v)
  Sets the user data for this widget.

• void visible () const
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

• void visible_focus (int v)
  Modifies keyboard focus navigation.

• int visible_r () const
  Returns whether a widget and all its parents are visible.

• int w () const
  Gets the widget width.

• Fl_When when () const
  Returns the conditions under which the callback is called.

• void when (uchar i)
  Sets the flags used to decide when a callback is called.

• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  Gets the widget position in its window.

• int y () const
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  Destroys the widget.

Protected Member Functions

• Fl_Cursor cursor (int n)
  Returns the cursor for cursor index n.

• int on_insert (Fl_Widget *, int) FL_OVERRIDE
  Insert a new entry in the size range list.

• int on_move (int, int) FL_OVERRIDE
  Move the entry in the size range list.

• void on_remove (int) FL_OVERRIDE
  Remove the entry from the size range list.

• void request_grow_b (int old_b, int &new_b, Fl_Rect *final_size)
  Request for children to change their layout.

• void request_grow_l (int old_l, int &new_l, Fl_Rect *final_size)
  Request for children to change their layout.

• void request_grow_r (int old_r, int &new_r, Fl_Rect *final_size)
  Request for children to change their layout.

• void request_grow_t (int old_t, int &new_t, Fl_Rect *final_size)
  Request for children to change their layout.

• void request_shrink_b (int old_b, int &new_b, Fl_Rect *final_size)
  Request for children to change their layout.
• void request_shrink_l (int old_l, int &new_l, Fl_Rect ∗final_size)
  Request for children to change their layout.
• void request_shrink_r (int old_r, int &new_r, Fl_Rect ∗final_size)
  Request for children to change their layout.
• void request_shrink_t (int old_t, int &new_t, Fl_Rect ∗final_size)
  Request for children to change their layout.
• void set_cursor (int n)
  Set one of four cursors used for dragging etc…

Protected Member Functions inherited from Fl_Group

• Fl_Rect ∗bounds ()
  Returns the internal array of widget sizes and positions.
• void draw () FL_OVERRIDE
  Draws the widget.
• void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.
• void draw_children ()
  Draws all children of the group.
• void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.
• int ∗sizes ()
  Returns the internal array of widget sizes and positions.
• void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const
  Draws the widget box according its box style.
• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
• void draw_focus () const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
• void draw_label () const
  Draws the widget’s label at the defined label position.
• void draw_label (int, int, int) const
  Draws the label in an arbitrary bounding box.
• Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.
• unsigned int flags () const
gets the widget flags mask.

- void h (int v)
  Internal use only.
- void set_flag (unsigned int c)
  Sets a flag in the flags mask.
- void w (int v)
  Internal use only.
- void x (int v)
  Internal use only.
- void y (int v)
  Internal use only.

Protected Attributes

- int cursor_
  current cursor index (0..3)
- Fl_Cursor * cursors_
  points at the array of 4 cursors (may be overridden)
- int default_min_h_
- int default_min_w_
- Size_Range * size_range_
- int size_range_capacity_
- int size_range_size_

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()
  Returns the currently active group.
- static void current (Fl_Group *g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
- static unsigned int labelShortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int testShortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
    INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
    FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
    OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
    MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
    GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
    NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
    AUTO_DELETE_USER_DATA = 1<<23 ,
    MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }

Generated by Doxygen
12.146 Fl_Tile Class Reference

12.146.1 Detailed Description

The Fl_Tile class lets you resize its children by dragging the border between them.

For the tiling to work correctly, the children of an Fl_Tile must cover the entire area of the widget, but must not overlap. This means that all children must touch each other at their edges, and no gaps can be left inside the Fl_Tile.

Fl_Tile does not normally draw any graphics of its own. The "borders" which can be seen in the snapshot above are actually part of the children. Their boxtypes have been set to FL_DOWN_BOX creating the impression of "ridges" where the boxes touch. What you see are actually two adjacent FL_DOWN_BOX's drawn next to each other. All neighboring widgets share the same edge - the widget's thick borders make it appear as though the widgets aren't actually touching, but they are. If the edges of adjacent widgets do not touch, then it will be impossible to drag the corresponding edges.

Note

Fl_Tile works in two distinctive modes. In classic mode, the range of motion for edges and intersections is controlled using an invisible child that is marked as the resizable() widget of the tile group. Classic mode is described in detail a few paragraphs down.

Fl_Tile size_range mode

By assigning a default minimum size to all children with Fl_Tile::init_size_range(int default_minimum_width, int default_minimum_height) or by assigning minimal sizes to individual children with Fl_Tile::size_range(Fl_Widget *child, int minimum_width, int minimum_height), the tile group is put into size_range operation mode.

In this mode, the child that is marked resizable() will behave as it would in a regular Fl_Group widget. When dragging edges or intersections with the mouse, Fl_Tile will ensure that none of the children shrinks to a size that is smaller than requested. When resizing the Fl_Tile group, size ranges are not enforced by the tile. Instead, the size range of the enclosing window should be limited to a valid range.

Tile does not differentiate between visible and invisible children. If children are created smaller than their assigned minimum size, dragging intersections may cause unexpected jumps in size. Zero width or height widget are not harmful, but should be avoided.

Example for a center document tile and two tool boxes on the left and right

```plaintext
Fl_Window win(400, 300, "My App");
Fl_Tile tile(0, 0, 400, 300);
Fl_Box left_tool_box(0, 0, 100, 300, "Tools");
left_tool_box.box(FL_DOWN_BOX);
tile.size_range(&left_tool_box, 50, 50);
Fl_Box document(100, 0, 200, 300, "Document");
document.box(FL_DOWN_BOX);
tile.size_range(&document, 100, 50);
Fl_Box right_tool_box(300, 0, 100, 300, "More\nTools");
right_tool_box.box(FL_DOWN_BOX);
```
tile.size_range(right_tool_box, 50, 50);
tile.end();
tile.resizeable(document);

win.end();
win.resizeable(tile);
win.show(argc, argv);
win.size_range(200, 50);

**Fl_Tile classic mode**

*Fl_Tile* allows objects to be resized to zero dimensions. To prevent this you can use the `resizeable()` to limit where corners can be dragged to. For more information see note below.

Even though objects can be resized to zero sizes, they must initially have non-zero sizes so the *Fl_Tile* can figure out their layout. If desired, call `position()` after creating the children but before displaying the window to set the borders where you want.

**Note on `resizeable(Fl_Widget &w)`:** The “resizeable” child widget (which should be invisible) limits where the borders can be dragged to. All dragging will be limited inside the resizeable widget’s borders. If you don’t set it, it will be possible to drag the borders right to the edges of the *Fl_Tile* widget, and thus resize objects on the edges to zero width or height. When the entire *Fl_Tile* widget is resized, the `resizeable()` widget will keep its border distance to all borders the same (this is normal resize behavior), so that you can effectively set a border width that will never change. To ensure correct event delivery to all child widgets the `resizeable()` widget must be the first child of the *Fl_Tile* widget group. Otherwise some events (e.g. FL_MOVE and FL_ENTER) might be consumed by the `resizeable()` widget so that they are lost for widgets covered (overlapped) by the `resizeable()` widget.

**Note**

You can still resize widgets inside the `resizeable()` to zero width and/or height, i.e. box 2b above to zero width and box 3a to zero height.

See also

```cpp
void Fl_Group::resizeable(Fl_Widget &w)
```

**Example for resizeable with 20 pixel border distance:**

```cpp
int dx = 20, dy = dx;
Fl_Tile tile(50, 50, 300, 300);
Fl_Box r(tile.x()+dx,tile.y()+dy,tile.w()-2*dx,tile.h()-2*dy);
tile.resizeable(r);
// ... create widgets inside tile (see test/tile.cxx) ...
tile.end();
```

See also the complete example program in test/tile.cxx.

## 12.146.2 Constructor & Destructor Documentation

### 12.146.2.1 Fl_Tile()

```cpp
Fl_Tile::Fl_Tile ( int X, int Y, int W, int H, const char * L = 0 )
```

Creates a new *Fl_Tile* widget using the given position, size, and label string. The default boxtype is FL_NO_BOX.

The destructor also deletes all the children. This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the *Fl_Tile* and all of its children can be automatic (local) variables, but you must declare the *Fl_Tile* first, so that it is destroyed last.

See also

- class Fl_Group
12.146.3 Member Function Documentation

12.146.3.1 cursor()

Fl_Cursor Fl_Tile::cursor ( int n ) [inline], [protected]

Returns the cursor for cursor index n.

See also

Fl_Tile::set_cursor(int)

12.146.3.2 drag_intersection()

void Fl_Tile::drag_intersection ( int oldx, int oldy, int newx, int newy ) [virtual]

Drags the intersection at (oldx,oldy) to (newx,newy).

See also

Fl_Tile::move_intersection(int oldx, int oldy, int newx, int newy) , but this method does not call init_sizes() and is used for interactive children layout using the mouse.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>oldx,oldy</th>
<th>move the intersection at this coordinate, pass zero to disable drag in that direction.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>newx,newy</td>
<td>move the intersection as close to this new coordinate as possible</td>
</tr>
</tbody>
</table>

12.146.3.3 handle()

int Fl_Tile::handle ( int event ) [virtual]

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class.

In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in    | event | the kind of event received |

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>if the event was not used or understood</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>
12.146.3.4 init_size_range()

```c
void Fl_Tile::init_size_range (  
    int default_min_w = -1,  
    int default_min_h = -1 )
```

Initialize the size range mode of Fl_Tile and set the default minimum width and height. The default minimum width and height is the size of the mouse pointer grab area at about 4 pixel units.

**Parameters**

- `default_min_w`: default minimum width
- `default_min_h`: default minimum height

12.146.3.5 move_intersection()

```c
void Fl_Tile::move_intersection (  
    int oldx,  
    int oldy,  
    int newx,  
    int newy ) [virtual]
```

Drags the intersection at `(oldx,oldy)` to `(newx,newy)`. This redraws all the necessary children.

If no size ranges are set, the new intersection position is limited to the size of the tile group. The `resizable()` option is not taken into account here.

If size ranges are set, the actual new position of the intersection will depend on the size range of every individual child. No child will be smaller than their `minw` and `minh`. After the new position is found, `move_intersection()` will call `init_sizes()`. The `resizable()` range is ignored.

**Parameters**

- `oldx(oldy)`: old intersection coordinates
- `newx(newy)`: new intersection coordinates

12.146.3.6 on_insert()

```c
int Fl_Tile::on_insert (  
    Fl_Widget * candidate,  
    int index ) [protected], [virtual]
```

Insert a new entry in the size range list.

Reimplemented from Fl_Group.

12.146.3.7 on_move()

```c
int Fl_Tile::on_move (  
    int oldIndex,  
    int newIndex ) [protected], [virtual]
```

Move the entry in the size range list.

Reimplemented from Fl_Group.

12.146.3.8 on_remove()

```c
void Fl_Tile::on_remove (  
```

See also

- Fl_Event

Reimplemented from Fl_Group.
int index ) [protected], [virtual]
Remove the entry from the size range list.
Reimplemented from Fl_Group.

12.146.3.9 position()

void Fl_Tile::position (  
    int oldx,  
    int oldy,  
    int newx,  
    int newy ) [inline]

 Deprecated "in 1.4.0 - use move_intersection(p) instead"

12.146.3.10 request_grow_b()

void Fl_Tile::request_grow_b (  
    int old_b,  
    int & new_b,  
    Fl_Rect * final_size ) [protected]
Request for children to change their layout.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>old_b</td>
</tr>
<tr>
<td>in,out</td>
<td>new_b</td>
</tr>
<tr>
<td>in,out</td>
<td>final_size</td>
</tr>
</tbody>
</table>

12.146.3.11 request_grow_l()

void Fl_Tile::request_grow_l (  
    int old_l,  
    int & new_l,  
    Fl_Rect * final_size ) [protected]
Request for children to change their layout.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>old_l</td>
</tr>
<tr>
<td>in,out</td>
<td>new_l</td>
</tr>
<tr>
<td>in,out</td>
<td>final_size</td>
</tr>
</tbody>
</table>

12.146.3.12 request_grow_r()

void Fl_Tile::request_grow_r (  
    int old_r,  
    int & new_r,  
    Fl_Rect * final_size ) [protected]
Request for children to change their layout.
12.146.3.13 request_grow_t()

```cpp
void Fl_Tile::request_grow_t ( int old_r, int &new_r, Fl_Rect *final_size ) [protected]
```

Request for children to change their layout.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>old_r</td>
<td>grow all children with this current right edge toward the right edge of this tile</td>
</tr>
<tr>
<td>in,out</td>
<td>new_r</td>
<td>try to grow to this coordinate, return the maximum possible growth (currently maxw is ignored, so we always grow to new_r)</td>
</tr>
<tr>
<td>in,out</td>
<td>final_size</td>
<td>write the new position and size of all affected children into this list of Fl_Rect</td>
</tr>
</tbody>
</table>

12.146.3.14 request_shrink_b()

```cpp
void Fl_Tile::request_shrink_b ( int old_b, int &new_b, Fl_Rect *final_size ) [protected]
```

Request for children to change their layout.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>old_b</td>
<td>shrink all children with this current bottom edge toward the top edge of this tile</td>
</tr>
<tr>
<td>in,out</td>
<td>new_b</td>
<td>try to shrink to this coordinate, return the maximum possible shrinkage</td>
</tr>
<tr>
<td>in,out</td>
<td>final_size</td>
<td>If not NULL, write the new position and size of all affected children into this list of Fl_Rect</td>
</tr>
</tbody>
</table>

12.146.3.15 request_shrink_l()

```cpp
void Fl_Tile::request_shrink_l ( int old_l, int &new_l, Fl_Rect *final_size ) [protected]
```

Request for children to change their layout.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>old_l</td>
<td>shrink all children with the left edge at old_l to shrink to new_l towards the right side of the tile. If the child can not shrink by that amount, it will ask all other children that touch its right side to shrink by the remainder (recursion). new_l will return the the maximum possible value while maintaining minimum width for all children involved.</td>
</tr>
<tr>
<td>in,out</td>
<td>new_l</td>
<td>request_shrink_r asks children to shrink toward the left, so that their right edge is as close as possible to new_r. request_shrink_t and request_shrink_b provide the same functionality for vertical resizing.</td>
</tr>
<tr>
<td>in,out</td>
<td>final_size</td>
<td>If not NULL, write the new position and size of all affected children into this list of Fl_Rect</td>
</tr>
</tbody>
</table>

See also

- Fl_Tile::request_shrink_l(int old_l, int &new_l, Fl_Rect *final_size)
12.146.3.16  request_shrink_r()

```c
void Fl_Tile::request_shrink_r (  
    int old_r,  
    int & new_r,  
    Fl_Rect * final_size ) [protected]
```

Request for children to change their layout.

See also

```c
Fl_Tile::request_shrink_l(int old_l, int &new_l, Fl_Rect *final_size)
```

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>old_r</th>
<th>shrink all children with this current right edge toward the left edge of this tile</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>new_r</td>
<td>try to shrink to this coordinate, return the maximum possible shrinkage</td>
</tr>
<tr>
<td>in,out</td>
<td>final_size</td>
<td>if not NULL, write the new position and size of all affected children into this list of Fl_Rect</td>
</tr>
</tbody>
</table>

12.146.3.17  request_shrink_t()

```c
void Fl_Tile::request_shrink_t (  
    int old_t,  
    int & new_t,  
    Fl_Rect * final_size ) [protected]
```

Request for children to change their layout.

See also

```c
Fl_Tile::request_shrink_l(int old_l, int &new_l, Fl_Rect *final_size)
```

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>old_t</th>
<th>shrink all children with this current top edge toward the bottom edge of this tile</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>new_t</td>
<td>try to shrink to this coordinate, return the maximum possible shrinkage</td>
</tr>
<tr>
<td>in,out</td>
<td>final_size</td>
<td>if not NULL, write the new position and size of all affected children into this list of Fl_Rect</td>
</tr>
</tbody>
</table>

12.146.3.18  resize()

```c
void Fl_Tile::resize (  
    int X,  
    int Y,  
    int W,  
    int H ) [virtual]
```

Resizes the Fl_Tile widget and its children.

Fl_Tile implements its own resize() method. It does not use Fl_Group::resize() to resize itself and its children. In size_range mode, the child marked resizable() is resized first. Only if its minimum size is reached, other widgets in the tile will resize too.
In classic mode or when no resizable child is set, enlarging works by moving the lower-right corner and resizing the bottom and right border widgets accordingly. Shrinking the Fl_Tile works in the opposite way by shrinking the bottom and right border widgets, unless they are reduced to zero width or height, resp. or to their minimal sizes defined by the resizable() widget. In this case other widgets will be shrunk as well.

See the Fl_Tile class documentation about how the resizable() works.
Reimplemented from Fl_Group.

12.146.3.19 set_cursor()

```cpp
void Fl_Tile::set_cursor (int n) [protected]
```

Set one of four cursors used for dragging etc... Fl_Tile uses an array of four cursors that are set depending on user actions:

- 0: normal cursor
- 1: horizontal dragging
- 2: vertical dragging
- 3: dragging an intersection

This method sets the window cursor for the given index n.

12.146.3.20 size_range() [1/2]

```cpp
void Fl_Tile::size_range (Fl_Widget * w, int minw, int minh, int maxw = 0x7FFFFFFF, int maxh = 0x7FFFFFFF)
```

Set the allowed size range for the give child widget. Fl_Tile currently supports only the minimal width and height setting.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>w</th>
<th>set the range for this child widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>minw,minh</td>
<td>minimum width and height for that child</td>
</tr>
<tr>
<td>in</td>
<td>maxw,maxh</td>
<td>maximum size, defaults to infinite, currently ignored</td>
</tr>
</tbody>
</table>

12.146.3.21 size_range() [2/2]

```cpp
void Fl_Tile::size_range (int index, int minw, int minh, int maxw = 0x7FFFFFFF, int maxh = 0x7FFFFFFF)
```

Set the allowed size range for the child at the given index. Fl_Tile currently supports only the minimal width and height setting.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>index</th>
<th>set the range for the child at this index</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>minw,minh</td>
<td>minimum width and height for that child</td>
</tr>
<tr>
<td>in</td>
<td>maxw,maxh</td>
<td>maximum size, defaults to infinite, currently ignored</td>
</tr>
</tbody>
</table>
The documentation for this class was generated from the following files:

- FL_Tile.H
- FL_Tile.cxx

### 12.147 FL_Tiled_Image Class Reference

This class supports tiling of images over a specified area.

```cpp
#include <Fl_Tiled_Image.H>
```

#### Inheritance diagram for Fl_Tiled_Image:

```
Fl_Tiled_Image
\|   \|   \|   \|
|   |   |   |   |
|   Fl_Image       |
|   \|   \|   \|   |
|       \|   \|   |
|       |   Fl_Image
```

#### Public Member Functions

- **void color_average (FL_Color c, float i) FL_OVERRIDE**
  
  The color_average() method averages the colors in the image with the provided FLTK color value.

- **Fl_Image * copy () const**

- **Fl_Image * copy (int W, int H) const FL_OVERRIDE**
  
  Creates a resized copy of the image.

- **void desaturate () FL_OVERRIDE**
  
  The desaturate() method converts an image to grayscale.

- **void draw (int X, int Y)**

- **void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE**
  
  Draws a tiled image.

- **Fl_Tiled_Image (Fl_Image *i, int W=0, int H=0)**
  
  The constructors create a new tiled image containing the specified image.

- **Fl_Image * image ()**
  
  Gets the image that is tiled.

- **virtual ~Fl_Tiled_Image ()**
  
  The destructor frees all memory and server resources that are used by the tiled image.

#### Public Member Functions inherited from Fl_Image

- **virtual class Fl_Shared_Image * as_shared_image ()**
  
  Returns whether an image is an Fl_Shared_Image or not.

- **Fl_Image * copy () const**
  
  Creates a copy of the image in the same size.

- **int count () const**
  
  Returns the number of data values associated with the image.

- **int d () const**
  
  Returns the image depth.

- **const char *const * data () const**
  
  Returns a pointer to the current image data array.

- **int data_h () const**
  
  Returns the height of the image data.

- **int data_w () const**
  
  Returns the width of the image data.

- **void draw (int X, int Y)**

Generated by Doxygen
Draws the image to the current drawing surface.

- **int fail () const**
  Returns a value that is not 0 if there is currently no image available.

- **Fl_Image (int W, int H, int D)**
  The constructor creates an empty image with the specified width, height, and depth.

- **int h () const**
  Returns the current image drawing height in FLTK units.

- **void inactive ()**
  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

- **virtual void label (Fl_Menu_Item ∗m)**
  This method is an obsolete way to set the image attribute of a menu item.

- **virtual void label (Fl_Widget ∗w)**
  This method is an obsolete way to set the image attribute of a widget or menu item.

- **int ld () const**
  Returns the current line data size in bytes.

- **virtual void release ()**
  Releases an Fl Image - the same as 'delete this'.

- **virtual void scale (int width, int height, int proportional=1, int can_expand=0)**
  Sets the drawing size of the image.

- **virtual void uncache ()**
  If the image has been cached for display, delete the cache data.

- **int w () const**
  Returns the current image drawing width in FLTK units.

- **virtual ~Fl_Image ()**
  The destructor is a virtual method that frees all memory used by the image.

### Protected Attributes

- **int alloc_image_**
- **Fl_Image ∗image_**

### Additional Inherited Members

#### Static Public Member Functions inherited from Fl_Image

- **static Fl_Labeltype define_FL_IMAGE_LABEL ()**
- **static Fl_RGB_Scaling RGB_scaling ()**
  Returns the currently used RGB image scaling method.

- **static void RGB_scaling (Fl_RGB_Scaling)**
  Sets the RGB image scaling method used for copy(int, int).

- **static Fl_RGB_Scaling scaling_algorithm ()**
  Gets what algorithm is used when resizing a source image to draw it.

- **static void scaling_algorithm (Fl_RGB_Scaling algorithm)**
  Sets what algorithm is used when resizing a source image to draw it.

#### Static Public Attributes inherited from Fl_Image

- **static const int ERR_FILE_ACCESS = -2**
- **static const int ERR_FORMAT = -3**
- **static const int ERR_MEMORY_ACCESS = -4**
- **static const int ERR_NO_IMAGE = -1**
- **static bool register_images_done = false**

  True after fl_register_images() was called, false before.
Protected Member Functions inherited from Fl_Image

- void d (int D)
  Sets the current image depth.
- void data (const char *const *p, int c)
  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)
  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)
  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)
  Sets the height of the image data.
- void ld (int LD)
  Sets the current line data size in bytes.
- void w (int W)
  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void labeltype (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void measure (const Fl_Label *lo, int &lw, int &lh)

12.147.1 Detailed Description
This class supports tiling of images over a specified area. The source (tile) image is not copied unless you call the color_average(), desaturate(), or inactive() methods.

12.147.2 Constructor & Destructor Documentation

12.147.2.1 Fl_Tiled_Image()

Fl_Tiled_Image::Fl_Tiled_Image ( Fl_Image *i, int W = 0, int H = 0 )
The constructors create a new tiled image containing the specified image. Use a width and height of 0 to tile the whole window/widget.

Note
Due to implementation constraints in FLTK 1.3.3 and later width and height of 0 may not work as expected when used as background image in widgets other than windows. You may need to center and clip the image (label) and set the label type to FL_NORMAL_LABEL. Doing so will let the tiled image fill the whole widget as its background image. Other combinations of label flags may or may not work.

#include "bg.xpm"
Fl_Pixmap *bg_xpm = new Fl_Pixmap(bg_xpm);
Fl_Tiled_Image *bg_tiled = new Fl_Tiled_Image(bg_xpm,0,0);
Fl_Box *box = new Fl_Box(40,40,300,100,"");
box->box(FL_UP_BOX);
box->labeltype(FL_NORMAL_LABEL);
box->align(FL_ALIGN_INSIDE | FL_ALIGN_CENTER | FL_ALIGN_CLIP);
box->image(bg_tiled);

Note
Setting an image (label) for a window may not work as expected due to implementation constraints in FLTK 1.3.x and maybe later. The reason is the way Fl::scheme() initializes the window's label type and image. A possible workaround is to use another Fl_Group as the only child widget and to set the background image for this group as described above.

Todo
Fix Fl_Tiled_Image as background image for widgets and windows and fix the implementation of Fl::scheme(const char *).
12.147.3 Member Function Documentation

12.147.3.1 color_average()

```cpp
void Fl_Tiled_Image::color_average ( 
    Fl_Color c,
    float i ) [virtual]
```

The color_average() method averages the colors in the image with the provided FLTK color value. The first argument specifies the FLTK color to be used. The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color. An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.

Reimplemented from Fl_Image.

12.147.3.2 copy()

```cpp
Fl_Image * Fl_Tiled_Image::copy ( 
    int W,
    int H ) const [virtual]
```

Creates a resized copy of the image. The new image should be released when you are done with it.

Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator delete for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's. The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().

For the new image the following equations are true:

- \( w() == data_w() == W \)
- \( h() == data_h() == H \)

Parameters

<table>
<thead>
<tr>
<th></th>
<th>W/H</th>
<th>Requested width and height of the new image</th>
</tr>
</thead>
</table>

Note

The returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_RGB_Image is an object of class Fl_Pixmap.

Since FLTK 1.4.0 this method is ‘const’. If you derive your own class from Fl_Image or any subclass your overridden methods of ‘Fl_Image::copy() const’ and ‘Fl_Image::copy(int, int) const’ must also be ‘const’ for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been ‘const’.

Reimplemented from Fl_Image.

12.147.3.3 desaturate()

```cpp
void Fl_Tiled_Image::desaturate ( ) [virtual]
```

The desaturate() method converts an image to grayscale. If the image contains an alpha channel (depth = 4), the alpha channel is preserved. An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.

Reimplemented from Fl_Image.
12.147.3.4 draw()

void Fl_Tiled_Image::draw (  
    int X,  
    int Y,  
    int W,  
    int H,  
    int cx = 0,  
    int cy = 0 ) [virtual]  

Draws a tiled image.
Tiled images can be used as background images for widgets and windows. However, due to implementation con-
straints, you must take care when setting label types and alignment flags. Only certain combinations work as
expected, others may yield unexpected results and undefined behavior.

This draw method can draw multiple copies of one image in an area given by X, Y, W, H.
The optional arguments cx and cy can be used to crop the image starting at offsets (cx, cy). cx and cy must be
\geq 0 (negative values are ignored). If one of the values is greater than the image width or height resp. (cx \geq
image()->w() or cy \geq image()->h()) nothing is drawn, because the resulting image would be empty.

After calculating the resulting image size the image is drawn as often as necessary to fill the given area, starting at
the top left corner.
If both W and H are 0 the image is repeated as often as necessary to fill the entire window, unless there is a valid
clip region. If you want to fill only one particular widget's background, then you should either set a clip region in your
draw() method or use the label alignment flags FL_ALIGN_INSIDE|FL_ALIGN_CLIP to make sure the image
is clipped.
This may be improved in a later version of the library.
Reimplemented from Fl_Image.
The documentation for this class was generated from the following files:

- Fl_Tiled_Image.H
- Fl_Tiled_Image.cxx

12.148 Fl_Timeout Class Reference

The internal class Fl_Timeout handles all timeout related functions.
#include <Fl_Timeout.h>

Static Public Member Functions

- static void add_timeout (double time, Fl_Timeout_Handler cb, void *data)
  Adds a one-shot timeout callback.
- static void do_timeouts ()
  Elapse timers and call their callbacks if any timers are expired.
- static void elapse_timeouts ()
  Elapse all timers w/o calling their callbacks.
- static int has_timeout (Fl_Timeout_Handler cb, void *data)
  Returns true if the timeout exists and has not been called yet.
- static void remove_timeout (Fl_Timeout_Handler cb, void *data)
  Remove a timeout callback.
- static void repeat_timeout (double time, Fl_Timeout_Handler cb, void *data)
  Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.
- static double time_to_wait (double tw)
  Returns the delay in seconds until the next timer expires, limited by ttw.

Protected Member Functions

- double delay ()
  Get the timer's delay in seconds.
• void delay (double t)
  Set the timer's delay in seconds.
• void insert ()
  Insert this timer entry into the active timer queue.
• void make_current ()
  Remove the timeout from the active timer queue and push it onto the stack of currently running callbacks.
• void release ()
  Remove the top-most timeout from the stack of currently running timeout callbacks and insert it into the list of free timers.

Static Protected Member Functions

• static Fl_Timeout * current ()
  Returns the first (top-most) timeout from the current timeout stack.
• static Fl_Timeout * get (double time, Fl_Timeout_Handler cb, void *data)
  Get an Fl_Timeout instance for further handling.

Protected Attributes

• Fl_Timeout_Handler callback
• void * data
• Fl_Timeout * next
• int skip
• double time

Static Protected Attributes

• static Fl_Timeout * current_timeout = 0
  The list of current timeouts is used to store the timeout whose callback is called while the callback is executed.
• static Fl_Timeout * first_timeout = 0
  List of active timeouts.
• static Fl_Timeout * free_timeout = 0
  List of free timeouts after use.

12.148.1 Detailed Description

The internal class Fl_Timeout handles all timeout related functions. All code is platform independent except retrieving a timestamp which requires calling a system driver function and potentially results in different timer resolutions (from milliseconds to microseconds).
Related user documentation:

• Fl_Timeout_Handler
• Fl::add_timeout(double time, Fl_Timeout_Handler cb, void *data)
• Fl::repeat_timeout(double time, Fl_Timeout_Handler cb, void *data)
• Fl::has_timeout(Fl_Timeout_Handler cb, void *data)
• Fl::remove_timeout(Fl_Timeout_Handler cb, void *data)
12.148.2 Member Function Documentation

12.148.2.1 add_timeout()

```c
void Fl_Timeout::add_timeout (
    double time,
    Fl_Timeout_Handler cb,
    void * data ) [static]
```

Adds a one-shot timeout callback.

The callback function `cb` will be called by `Fl::wait()` at `time` seconds after this function is called.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>delta time in seconds until the timer expires</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cb</td>
<td>callback function</td>
</tr>
<tr>
<td>in</td>
<td>data</td>
<td>optional user data (default: NULL)</td>
</tr>
</tbody>
</table>

Implements `Fl::add_timeout(double time, Fl_Timeout_Handler cb, void *data)`

See also

`Fl::add_timeout(double time, Fl_Timeout_Handler cb, void *data)`

12.148.2.2 current()

`Fl_Timeout * Fl_Timeout::current ( ) [static], [protected]`

Returns the first (top-most) timeout from the current timeout stack. This returns a pointer to the timeout but does not remove it from the list of current timeouts. This should be the timeout that is currently executing its callback.

Returns

`Fl_Timeout*` The current timeout whose callback is running.

Return values

`NULL` if no callback is currently running.

12.148.2.3 elapse_timeouts()

`void Fl_Timeout::elapse_timeouts ( ) [static]`

Elapse all timers w/o calling their callbacks. All timer values are adjusted by the delta time since the last call. This method does NOT call timer callbacks if timers are expired. This must be called before new timers are added to the timer queue to make sure that the next timer decrement does not count down too much time.

See also

`Fl_Timeout::do_timeouts()`

12.148.2.4 get()

`Fl_Timeout * Fl_Timeout::get ( double time,
                                 Fl_Timeout_Handler cb,
                                 void * data ) [static], [protected]`

Get an `Fl_Timeout` instance for further handling.

The timer object will be initialized with the input parameters as given by `Fl::add_timeout()` or `Fl::repeat_timeout()`. `Fl_Timeout` objects are maintained in three queues:

- active timer queue
- list (stack, i.e. LIFO) of currently executing timer callbacks
- free timer entries.

When the FLTK program is launched all queues are empty. Whenever a new timer object is required the `get()` method is called and a timer object is either found in the queue of free timer entries or a new timer object is created (operator new).
Active timer entries are inserted into the "active timer queue" until they expire and their callback is called. Before the callback is called the timer entry is inserted into the list of current timers, i.e. it becomes the \texttt{Fl\_Timeout::current()} timeout. This can be used in \texttt{Fl::repeat\_timeout()} to find out if and how long the current timeout has been delayed. When a timer is no longer used it is popped from the current list and inserted into the "free timer" list so it can be reused later. Timer queue entries are never returned to the system, there's no garbage collection. The total number of timer objects is determined by the largest number of concurrently active timers.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>requested delta time</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>\texttt{cb}</td>
<td>timer callback</td>
</tr>
<tr>
<td>in</td>
<td>\texttt{data}</td>
<td>userdata for timer callback</td>
</tr>
</tbody>
</table>

### Returns

\texttt{Fl\_Timeout\*} Timer entry

### See also

\texttt{Fl::add\_timeout()}, \texttt{Fl::repeat\_timeout()}

#### 12.148.2.5 \texttt{has\_timeout()}

```c
int Fl\_Timeout::has\_timeout (  
    Fl\_Timeout\_Handler \texttt{cb},  
    void * \texttt{data} ) \texttt{[static]}
```

Returns true if the timeout exists and has not been called yet.

### Parameters

<table>
<thead>
<tr>
<th>in \texttt{cb}</th>
<th>Timer callback (must match)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in \texttt{data}</td>
<td>Callback user data (must match)</td>
</tr>
</tbody>
</table>

### Returns

whether the timer was found in the queue

### Return values

<table>
<thead>
<tr>
<th>0</th>
<th>not found</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>found</td>
</tr>
</tbody>
</table>

Implements \texttt{Fl::has\_timeout(Fl\_Timeout\_Handler cb, void *data)}

### See also

\texttt{Fl::has\_timeout(Fl\_Timeout\_Handler cb, void *data)}

#### 12.148.2.6 \texttt{insert()}

```c
void Fl\_Timeout::insert ( ) \texttt{[protected]}
```

Insert this timer entry into the active timer queue. The timer is inserted at the required position so the timer queue is always ordered by due time.
12.148.2.7 make_current()

```cpp
void Fl_Timeout::make_current ( ) [protected]
```

Remove the timeout from the active timer queue and push it onto the stack of currently running callbacks.
This becomes the current() timeout which can be used in Fl::repeat_timeout().

See also

```cpp
Fl_Timeout::current()
```

12.148.2.8 release()

```cpp
void Fl_Timeout::release ( ) [protected]
```

Remove the top-most timeout from the stack of currently running timeout callbacks and insert it into the list of free
timers.

Typical code in the library would look like:

```cpp
// The timeout \p t has expired, run its callback
(t->callback)(t->data);
```

12.148.2.9 remove_timeout()

```cpp
void Fl_Timeout::remove_timeout (  
    Fl_Timeout_Handler cb,  
    void * data ) [static]
```

Remove a timeout callback.
This method removes all matching timeouts, not just the first one. This may change in the future.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cb</th>
<th>Timer callback to be removed (must match)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>Wildcard if NULL, must match otherwise</td>
</tr>
</tbody>
</table>

Implements Fl::remove_timeout(Fl_Timeout_Handler cb, void *data)

See also

```cpp
Fl::remove_timeout(Fl_Timeout_Handler cb, void *data)
```

12.148.2.10 repeat_timeout()

```cpp
void Fl_Timeout::repeat_timeout (  
    double time,  
    Fl_Timeout_Handler cb,  
    void * data ) [static]
```

Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>delta time in seconds until the timer expires</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cb</td>
<td>callback function</td>
</tr>
<tr>
<td>in</td>
<td>data</td>
<td>optional user data (default: NULL)</td>
</tr>
</tbody>
</table>

Implements Fl::repeat_timeout(double time, Fl_Timeout_Handler cb, void *data)

See also

```cpp
Fl::repeat_timeout(double time, Fl_Timeout_Handler cb, void *data)
```
12.148.2.11  time_to_wait()

double Fl_Timeout::time_to_wait (  
    double ttw  ) [static]

Returns the delay in seconds until the next timer expires, limited by \( ttw \).
This function calculates the time to wait for the FLTK event queue processing, depending on the given value \( ttw \).
If at least one timer is active and its timeout value is smaller than \( ttw \) then this value is returned. \( Fl::wait() \) will wait no longer than until the next timer expires.
If no timer is active this returns the input value \( ttw \) unchanged.
If at least one timer is expired this returns 0.0 so the event processing does not wait.

Parameters

| in  | ttw | time to wait from \( Fl::wait() \) etc. (upper limit) |

Returns

delay until next timeout or 0.0 (see description)

12.148.3  Member Data Documentation

12.148.3.1  current_timeout

\( Fl::Timeout :: current_timeout = 0 \) [static], [protected]
The list of current timeouts is used to store the timeout whose callback is called while the callback is executed.
This is used like a stack, the current timeout is pushed to the front of the list and once the callback is finished, that timeout is removed and entered into the free list.
Background: \( Fl::repeat\_timeout() \) needs to know which timeout triggered it and the exact schedule time and/or the delay of that timeout, i.e. how long the scheduled time was missed before the callback was called. A static, global variable is not sufficient since the user code can call other functions, e.g. dialogs, that run a nested event loop which can run another timeout callback. Hence this list of "current" timeouts is used like a stack (last in, first out).

See also

\( Fl::Timeout :: push() \) Member function (method)

12.148.3.2  first_timeout

\( Fl::Timeout :: first\_timeout = 0 \) [static], [protected]
List of active timeouts.
These timeouts can be triggered when due, which calls their callbacks. The lifetime of a timeout:

- active, in this queue
- callback running, in queue \( current\_timeout \)
- done, in list of free timeouts, ready to be reused.

12.148.3.3  free_timeout

\( Fl::Timeout :: free\_timeout = 0 \) [static], [protected]
List of free timeouts after use.
Timeouts can be reused many times.
The documentation for this class was generated from the following files:

- \( Fl::Timeout.h \)
- \( Fl::Timeout.cpp \)
12.149 Fl_Timer Class Reference

This is provided only to emulate the Forms Timer widget.

```c
#include <Fl_Timer.H>
```

Inheritance diagram for Fl_Timer:

```
Fl_Widget
  ↓
Fl_Timer
```

Public Member Functions

- `char direction () const`
  
  Gets or sets the direction of the timer.

- `void direction (char d)`
  
  Gets or sets the direction of the timer.

- `Fl_Timer (uchar t, int x, int y, int w, int h, const char ∗l)`
  
  Creates a new Fl_Timer widget using the given type, position, size, and label string.

- `int handle (int) FL_OVERRIDE`
  
  Handles the specified event.

- `char suspended () const`
  
  Gets or sets whether the timer is suspended.

- `void suspended (char d)`
  
  Gets or sets whether the timer is suspended.

- `double value () const`
  
  
  See void Fl_Timer::value(double)

- `void value (double)`
  
  Sets the current timer value.

- `~Fl_Timer ()`
  
  Destroys the timer and removes the timeout.

Public Member Functions inherited from Fl_Widget

- `void _clear_fullscreen ()`

- `void _set_fullscreen ()`

- `void activate ()`
  
  Activates the widget.

- `unsigned int active () const`
  
  Returns whether the widget is active.

- `int active_r () const`
  
  Returns whether the widget and all of its parents are active.

- `Fl_Align align () const`
  
  Gets the label alignment.

- `int active_r () const`
  
  Returns whether the widget and all of its parents are active.

- `Fl_Align align () const`
  
  Gets the label alignment.

- `void align (Fl_Align alignment)`
  
  Sets the label alignment.

- `long argument () const`
  
  Gets the current user data (long) argument that is passed to the callback function.

- `void argument (long v)`
  
  Sets the current user data (long) argument that is passed to the callback function.

- `virtual class Fl_Gl_Window ∗ as_gl_window ()`
Returns an `Fl_Gl_Window` pointer if this widget is an `Fl_Gl_Window`.

- virtual class `Fl_Gl_Window` const * as_gl_window () const
- virtual `Fl_Group` const * as_group ()
  Returns an `Fl_Group` pointer if this widget is an `Fl_Group`.

- virtual `Fl_Group` const * as_group () const
- virtual `Fl_Window` const * as_window ()
  Returns an `Fl_Window` pointer if this widget is an `Fl_Window`.

- void bind_deimage (Fl_Image * img)
  Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image (Fl_Image * img)
  Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- `Fl_Boxtype` box () const
  Gets the box type of the widget.

- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.

- `Fl_Callback_p` callback () const
  Gets the current callback function for the widget.

- void callback (Fl_Callback * cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback * cb, Fl_Callback_User_Data * p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback * cb, void * p)
  Sets the current callback function and data for the widget.

- void callback (Fl_Callback0 * cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback1 * cb, long p=0)
  Sets the current callback function for the widget.

- unsigned int changed () const
  Checks if the widget value changed since the last callback.

- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  Clears or sets the damage flags.

- void clear_output ()
  Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.

- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

- `Fl_Color` color () const
  Gets the background color of the widget.

- void color (Fl_Color bg)
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
Sets the background and selection color of the widget.

- **FL_Color color2 () const**
  - For back compatibility only.
- **void color2 (unsigned a)**
  - For back compatibility only.
- **int contains (const FL_Widget *w) const**
  - Checks if w is a child of this widget.
- **void copy_label (const char *new_label)**
  - Sets the current label.
- **void copy_tooltip (const char *text)**
  - Sets the current tooltip text.
- **uchar damage () const**
  - Returns non-zero if draw() needs to be called.
- **void damage (uchar c)**
  - Sets the damage bits for the widget.
- **void damage (uchar c, int x, int y, int w, int h)**
  - Sets the damage bits for an area inside the widget.
- **int damage_resize (int, int, int, int)**
  - Internal use only.
- **void deactivate ()**
  - Deactivates the widget.
- **FL_Image *deimage ()**
  - Gets the image that is used as part of the widget label when in the inactive state.
- **const FL_Image *deimage () const**
  - Gets the image that is used as part of the widget label when in the inactive state.
- **void deimage (Fl_Image &img)**
  - Sets the image to use as part of the widget label when in the inactive state.
- **void deimage (Fl_Image *img)**
  - Sets the image to use as part of the widget label when in the inactive state.
- **int deimage_bound () const**
  - Returns whether the inactive image is managed by the widget.
- **void do_callback (FL_Callback_Reas one=FL_REASON_UNKNOWN)**
  - Calls the widget callback function with default arguments.
- **void do_callback (FL_Widget *widget, long arg, FL_Callback_Reas one=FL_REASON_UNKNOWN)**
  - Calls the widget callback function with arbitrary arguments.
- **void do_callback (FL_Widget *widget, void *arg=0, FL_Callback_Reas one=FL_REASON_UNKNOWN)**
  - Calls the widget callback function with arbitrary arguments.
- **void draw_label (int, int, int, int, FL_Align) const**
  - Draws the label in an arbitrary bounding box with an arbitrary alignment.
- **int h () const**
  - Gets the widget height.
- **virtual void hide ()**
  - Makes a widget invisible.
- **FL_Image *image ()**
  - Gets the image that is used as part of the widget label when in the active state.
- **const FL_Image *image () const**
  - Gets the image that is used as part of the widget label when in the active state.
- **void image (FL_Image &img)**
  - Sets the image to use as part of the widget label when in the active state.
- **void image (FL_Image *img)**
  - Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget ∗wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char ∗ label () const
  Gets the current label text.
• void label (const char ∗text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char ∗b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group ∗ parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group ∗p)
  Internal use only - "for hacks only".
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
Sets the selection color.

• void set_active ()
  Marks the widget as active without sending events or changing focus.

• void set_changed ()
  Marks the value of the widget as changed.

• void set_output ()
  Sets a widget to output only.

• void set_visible ()
  Makes the widget visible.

• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.

• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.

• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

• virtual void show ()
  Makes a widget visible.

• void size (int W, int H)
  Changes the size of the widget.

• int take_focus ()
  Gives the widget the keyboard focus.

• unsigned int takesevents () const
  Returns if the widget is able to take events.

• int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.

• const char ∗ tooltip () const
  Gets the current tooltip text.

• void tooltip (const char ∗text)
  Sets the current tooltip text.

• Fl_Window ∗ top_window () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const
  Gets the widget type.

• void type (uchar t)
  Sets the widget type.

• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void ∗ user_data () const
  Gets the user data for this widget.

• void user_data (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.

• void user_data (void ∗v)
  Sets the user data for this widget.

• unsigned int visible () const
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.

• int w () const
  Gets the widget width.

• Fl_When when () const
  Returns the conditions under which the callback is called.

• void when (uchar i)
  Sets the flags used to decide when a callback is called.

• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  Gets the widget position in its window.

• int y () const
  Gets the widget position in its window.

• virtual ~Fl_Widget()
  Destroys the widget.

Protected Member Functions

• void draw () FL_OVERRIDE
  Draws the widget.

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.

• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• void draw_box () const
  Draws the widget box according to its box style.

• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

• void draw_focus () const
  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

• void draw_label () const
  Draws the widget's label at the defined label position.

• void draw_label (int, int, int) const
  Draws the label in an arbitrary bounding box.

• Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.

• unsigned int flags () const
  Gets the widget flags mask.

• void h (int v)
  Internal use only.

• void set_flag (unsigned int c)
  Sets a flag in the flags mask.
• void \texttt{w} (int v)
  \textit{Internal use only.}
• void \texttt{x} (int v)
  \textit{Internal use only.}
• void \texttt{y} (int v)
  \textit{Internal use only.}

**Additional Inherited Members**

**Static Public Member Functions inherited from \texttt{Fl_Widget}**

• static void \texttt{default_callback} (\texttt{Fl_Widget} \ast widget, void \ast data)
  \textit{The default callback for all widgets that don't set a callback.}
• static unsigned int \texttt{label_shortcut} (const char \ast t)
  \textit{Returns the Unicode value of the '&x' shortcut in a given text.}
• static int \texttt{test_shortcut} (const char \ast t, const bool require_alt=false)
  \textit{Returns true if the given text \texttt{t} contains the entered '&x' shortcut.}

**Protected Types inherited from \texttt{Fl_Widget}**

• enum {
  INACTIVE = 1 << 0, INVISIBLE = 1 << 1, OUTPUT = 1 << 2, NOBORDER = 1 << 3,
  FORCE_POSITION = 1 << 4, NON_MODAL = 1 << 5, SHORTCUT_LABEL = 1 << 6, CHANGED = 1 << 7,
  OVERRIDE = 1 << 8, VISIBLE_FOCUS = 1 << 9, COPIED_LABEL = 1 << 10, CLIP_CHILDREN = 1 << 11,
  MENU_WINDOW = 1 << 12, TOOLTIP_WINDOW = 1 << 13, MODAL = 1 << 14, NO_OVERLAY = 1 << 15,
  GROUP_RELATIVE = 1 << 16, COPIED_TOOLTIP = 1 << 17, FULLSCREEN = 1 << 18, MAC_USE_ACCENTS_MENU
  = 1 << 19, NEEDS_KEYBOARD = 1 << 20, IMAGE_BOUND = 1 << 21, DEIMAGE_BOUND = 1 << 22,
  AUTO_DELETE_USER_DATA = 1 << 23, MAXIMIZED = 1 << 24, USERFLAG3 = 1 << 29, USERFLAG2 = 1 << 30, USERFLAG1 = 1 << 31 }
  \textit{flags possible values enumeration.}

12.149.1 Detailed Description

This is provided only to emulate the Forms Timer widget. It works by making a timeout callback every 1/5 second. This is wasteful and inaccurate if you just want something to happen a fixed time in the future. You should directly call \texttt{Fl::add_timeout()} instead.

12.149.2 Constructor & Destructor Documentation

12.149.2.1 \texttt{Fl_Timer()}

\begin{verbatim}
Fl_Timer::Fl_Timer ( 
  uchar t,
  int X,
  int Y,
  int W,
  int H,
  const char * l )
\end{verbatim}

Creates a new \texttt{Fl_Timer} widget using the given type, position, size, and label string. The type parameter can be any of the following symbolic constants:

• \texttt{FL_NORMAL_TIMER} - The timer just does the callback and displays the string "Timer" in the widget.
• \texttt{FL_VALUE_TIMER} - The timer does the callback and displays the current timer value in the widget.
• \texttt{FL_HIDDEN_TIMER} - The timer just does the callback and does not display anything.
### 12.149.3 Member Function Documentation

#### 12.149.3.1 direction() [1/2]

```cpp
char Fl_Timer::direction ( ) const [inline]
```

Gets or sets the direction of the timer.
If the direction is zero then the timer will count up, otherwise it will count down from the initial value().

#### 12.149.3.2 direction() [2/2]

```cpp
void Fl_Timer::direction (char d ) [inline]
```

Gets or sets the direction of the timer.
If the direction is zero then the timer will count up, otherwise it will count down from the initial value().

#### 12.149.3.3 draw()

```cpp
void Fl_Timer::draw ( ) [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.
Override this function to draw your own widgets.
If you ever need to call another widget's `draw` method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:
```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

#### 12.149.3.4 handle()

```cpp
int Fl_Timer::handle (int event ) [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class.
This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your `handle()` method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the `handle()` method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>event</th>
<th>the kind of event received</th>
</tr>
</thead>
</table>

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>if the event was not used or understood</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>

**See also**

- `Fl_Event`

Reimplemented from `Fl_Widget`.

Generated by Doxygen
12.149.3.5  suspended()

```cpp
char Fl_Timer::suspended ( ) const [inline]
```

Gets or sets whether the timer is suspended.

The documentation for this class was generated from the following files:

- FL_Timer.H
- forms_timer.cxx

12.150  Fl_Toggle_Button Class Reference

The toggle button is a push button that needs to be clicked once to toggle on, and one more time to toggle off.

```cpp
#include <Fl_Toggle_Button.H>
```

Inheritance diagram for Fl_Toggle_Button:

```
+----------------+          +-------------------+
| Fl_Toggle_Button|  ↓                |
|                 |  ↓                |
| Fl_Button       |  ↓                |
|                 |  ↓                |
| Fl_Widget       |
```

Public Member Functions

- **Fl_Toggle_Button (int X, int Y, int W, int H, const char ∗l=0)**
  
  Creates a new Fl_Toggle_Button widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Button

- **int clear ()**
  
  Same as value(0).

- **uchar compact ()**
  
  Return true if buttons are rendered as compact buttons.

- **void compact (uchar v)**
  
  Decide if buttons should be rendered in compact mode.

- **Fl_Boxtype down_box () const**
  
  Returns the current down box type, which is drawn when value() is non-zero.

- **void down_box (Fl_Boxtype b)**
  
  Sets the down box type.

- **Fl_Color down_color () const**
  
  (for backwards compatibility)

- **void down_color (unsigned c)**
  
  (for backwards compatibility)

- **Fl_Button (int X, int Y, int W, int H, const char ∗L=0)**
  
  The constructor creates the button using the given position, size, and label.

- **int handle (int) FL_OVERRIDE**
  
  Handles the specified event.

- **int set ()**
  
  Same as value(1).

- **void setonly ()**
  
  Turns on this button and turns off all other radio buttons in the group (calling value(1) or set () does not do this).
• int **shortcut** () const  
  Returns the current shortcut key for the button.

• void **shortcut** (const char *s)  
  (for backwards compatibility)

• void **shortcut** (int s)  
  Sets the shortcut key to s.

• char **value** () const  
  Returns the current value of the button (0 or 1).

• int **value** (int v)  
  Sets the current value of the button.

**Public Member Functions inherited from Fl_Widget**

• void **_clear_fullscreen** ()

• void **_set_fullscreen** ()

• void **activate** ()
  Activates the widget.

• unsigned int **active** () const  
  Returns whether the widget is active.

• int **active_r** () const  
  Returns whether the widget and all of its parents are active.

• Fl_Align **align** () const  
  Gets the label alignment.

• void **align** (Fl_Align alignment)  
  Sets the label alignment.

• long **argument** () const  
  Gets the current user data (long) argument that is passed to the callback function.

• void **argument** (long v)  
  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window * as_gl_window ()  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const * as_gl_window () const

• virtual Fl_Group * as_group ()  
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Group const * as_group () const

• virtual Fl_Window * as_window ()  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window const * as_window () const

• void **bind_deimage** (Fl_Image *img)  
  Sets the image to use as part of the widget label when in the inactive state.

• void **bind_deimage** (int f)  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

• void **bind_image** (Fl_Image *img)  
  Sets the image to use as part of the widget label when in the active state.

• void **bind_image** (int f)  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

• Fl_Boxtype **box** () const  
  Gets the box type of the widget.

• void **box** (Fl_Boxtype new_box)  
  Sets the box type for the widget.

• Fl_Callback_p **callback** () const

Generated by Doxygen
Gets the current callback function for the widget.

- void callback (Fl_Callback cb)
  - Sets the current callback function for the widget.

- void callback (Fl_Callback cb, Fl_Callback_User_Data p, bool auto_free)
  - Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback cb, void p)
  - Sets the current callback function and data for the widget.

- void callback (Fl_Callback cb)
  - Sets the current callback function for the widget.

- void callback (Fl_Callback cb, long p=0)
  - Sets the current callback function for the widget.

- unsigned int changed () const
  - Checks if the widget value changed since the last callback.

- void clear_active ()
  - Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  - Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  - Clears or sets the damage flags.

- void clear_output ()
  - Sets a widget to accept input.

- void clear_visible ()
  - Hides the widget.

- void clear_visible_focus ()
  - Disables keyboard focus navigation with this widget.

- void color (Fl_Color bg)
  - Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  - Sets the background and selection color of the widget.

- void color2 (unsigned a)
  - For back compatibility only.

- Fl_Color color () const
  - Gets the background color of the widget.

- void contains (const Fl_Widget w) const
  - Checks if w is a child of this widget.

- void copy_label (const char new_label)
  - Sets the current label.

- void copy_tooltip (const char text)
  - Sets the current tooltip text.

- uchar damage () const
  - Returns non-zero if draw() needs to be called.

- void damage (uchar c)
  - Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)
  - Sets the damage bits for an area inside the widget.

- int damage_resize (int, int, int, int)
  - Internal use only.

- void deactivate ()
  - Deactivates the widget.
Fl_Toggle_Button Class Reference

- FL_Image * deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

- const FL_Image * deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

- void deimage (FL_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

- void deimage (FL_Image img)
  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

- void do_callback (FL_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

- void do_callback (FL_Widget *widget, long arg, FL_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- void do_callback (FL_Widget *widget, void *arg=0, FL_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- int h () const
  Gets the widget height.

- virtual void hide ()
  Makes a widget invisible.

- FL_Image * image ()
  Gets the image that is used as part of the widget label when in the active state.

- const FL_Image * image () const
  Gets the image that is used as part of the widget label when in the active state.

- void image (FL_Image &img)
  Sets the image to use as part of the widget label when in the active state.

- void image (FL_Image img)
  Sets the image to use as part of the widget label when in the active state.

- int image_bound () const
  Returns whether the image is managed by the widget.

- int inside (const FL_Widget *wgt) const
  Checks if this widget is a child of wgt.

- int is_label_copied () const
  Returns whether the current label was assigned with copy_label().

- const char * label () const
  Gets the current label text.

- void label (const char *text)
  Sets the current label pointer.

- void label (FL_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.

- FL_Color labelcolor () const
  Gets the label color.

- void labelcolor (FL_Color c)
  Sets the label color.

- FL_Font labelfont () const
  Gets the font to use.

- void labelfont (FL_Font f)
  Sets the font to use.

- FL_Fontsize labelsize () const
  Gets the font size to use.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - “for hacks only”.
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int take_sevets () const
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered 'x' shortcut.
• const char * tooltip () const
  Gets the current tooltip text.
• void tooltip (const char *text)
  Sets the current tooltip text.
• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window * top_window_offset (int xoff, int yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.
Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)
  
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)
  
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

Protected Member Functions inherited from Fl_Button

- void draw () FL_OVERRIDE
  
  Draws the widget.
- void simulate_key_action ()

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  
  Clears a flag in the flags mask.
- void draw_backdrop () const
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h) const
  
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.
- void draw_label () const
  
  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const
12.150 Fl_Toggle_Button Class Reference

Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.

- unsigned int flags () const
  Gets the widget flags mask.

- void h (int v)
  *Internal use only.*

- void set_flag (unsigned int c)
  *Internal use only.*

- void w (int v)
  *Internal use only.*

- void x (int v)
  *Internal use only.*

- void y (int v)
  *Internal use only.*

Static Protected Member Functions inherited from Fl_Button

- static void key_release_timeout (void *)

Static Protected Attributes inherited from Fl_Button

- static Fl_Widget_Tracker * key_release_tracker = 0

12.150.1 Detailed Description

The toggle button is a push button that needs to be clicked once to toggle on, and one more time to toggle off. The Fl_Toggle_Button subclass displays the "on" state by drawing a pushed-in button. Buttons generate callbacks when they are clicked by the user. You control exactly when and how by changing the values for type() and when().

12.150.2 Constructor & Destructor Documentation

12.150.2.1 Fl_Toggle_Button()

Fl_Toggle_Button::Fl_Toggle_Button (int X, int Y, int W, int H, const char *L = 0L)

Creates a new Fl_Toggle_Button widget using the given position, size, and label string.

The constructor creates the button using the given position, size, and label.

The inherited destructor deletes the toggle button.

The Button type() is set to FL_TOGGLE_BUTTON.

Parameters

| In | X, Y, W, H | position and size of the widget |
| In | L | widget label, default is no label |

The documentation for this class was generated from the following files:

- Fl_Toggle_Button.H
- Fl_Button.cxx

Generated by Doxygen
The Fl_Tooltip class provides tooltip support for all FLTK widgets.

```c
#include <Fl_Tooltip.H>
```

### Static Public Member Functions

- **static Fl_Color color ()**
  
  Gets the background color for tooltips.

- **static void color (Fl_Color c)**
  
  Sets the background color for tooltips.

- **static Fl_Widget * current ()**
  
  Gets the current widget target.

- **static void current (Fl_Widget *)**
  
  Sets the current widget target.

- **static Fl_Window * current_window ()**
  
  Returns the window that is used for tooltips.

- **static float delay ()**
  
  Gets the tooltip delay.

- **static void delay (float f)**
  
  Sets the tooltip delay.

- **static void disable ()**
  
  Same as enable(0), disables tooltips on all widgets.

- **static void enable (int b=1)**
  
  Enables tooltips on all widgets (or disables if b is false).

- **static int enabled ()**
  
  Returns non-zero if tooltips are enabled.

- **static void enter_area (Fl_Widget *w, int X, int Y, int W, int H, const char *tip)**
  
  You may be able to use this to provide tooltips for internal pieces of your widget.

- **static Fl_Font font ()**
  
  Gets the typeface for the tooltip text.

- **static void font (Fl_Font i)**
  
  Sets the typeface for the tooltip text.

- **static float hidedelay ()**
  
  Gets the time until an open tooltip hides again.

- **static void hidedelay (float f)**
  
  Sets the time until an open tooltip hides again.

- **static float hoverdelay ()**
  
  Gets the tooltip hover delay, the delay between tooltips.

- **static void hoverdelay (float f)**
  
  Sets the tooltip hover delay, the delay between tooltips.

- **static int margin_height ()**
  
  Gets the amount of extra space above and below the tooltip’s text.

- **static void margin_height (int v)**
  
  Sets the amount of extra space above and below the tooltip’s text.

- **static int margin_width ()**
  
  Gets the amount of extra space left/right of the tooltip’s text.

- **static void margin_width (int v)**
  
  Sets the amount of extra space left/right of the tooltip’s text.

- **static Fl_Fontsize size ()**
  
  Gets the size of the tooltip text.

- **static void size (Fl_Fontsize s)**
Sets the size of the tooltip text.

- static Fl_Color textcolor ()

  Gets the color of the text in the tooltip.

- static void textcolor (Fl_Color c)

  Sets the color of the text in the tooltip.

- static int wrap_width ()

  Gets the maximum width for tooltip's text before it word wraps.

- static void wrap_width (int v)

  Sets the maximum width for tooltip's text before it word wraps.

Static Public Attributes

- static void(* enter)(Fl_Widget *w) = nothing
- static void(* exit)(Fl_Widget *w) = nothing

Friends

- class Fl_TooltipBox
- void Fl_Widget::copy_tooltip (const char *)
- void Fl_Widget::tooltip (const char *)

12.151.1 Detailed Description

The Fl_Tooltip class provides tooltip support for all FLTK widgets. It contains only static methods.

12.151.2 Member Function Documentation

12.151.2.1 color() [1/2]

static Fl_Color Fl_Tooltip::color () [inline], [static]
Gets the background color for tooltips.
The default background color is a pale yellow.

12.151.2.2 color() [2/2]

static void Fl_Tooltip::color (Fl_Color c) [inline], [static]
Sets the background color for tooltips.
The default background color is a pale yellow.
12.151.2.3 current()

void Fl_Tooltip::current ( 
    Fl_Widget * w ) [static]

Sets the current widget target.
Acts as though enter(widget) was done but does not pop up a tooltip. This is useful to prevent a tooltip from
reappearing when a modal overlapping window is deleted. FLTK does this automatically when you click the mouse
button.

12.151.2.4 delay() [1/2]

static float Fl_Tooltip::delay ( ) [inline], [static]

Gets the tooltip delay.
The default delay is 1.0 seconds.

12.151.2.5 delay() [2/2]

static void Fl_Tooltip::delay ( 
    float f ) [inline], [static]

Sets the tooltip delay.
The default delay is 1.0 seconds.

12.151.2.6 disable()

static void Fl_Tooltip::disable ( ) [inline], [static]

Same as enable(0), disables tooltips on all widgets.

12.151.2.7 enable()

static void Fl_Tooltip::enable ( 
    int b = 1 ) [inline], [static]

Enables tooltips on all widgets (or disables if b is false).

12.151.2.8 enabled()

static int Fl_Tooltip::enabled ( ) [inline], [static]

Returns non-zero if tooltips are enabled.

12.151.2.9 enter_area()

void Fl_Tooltip::enter_area ( 
    Fl_Widget * wid, 
    int x, 
    int y, 
    int w, 
    int h, 
    const char * t ) [static]

You may be able to use this to provide tooltips for internal pieces of your widget.
Call this after setting Fl::belowmouse() to your widget (because that calls the above enter() method). Then figure
out what thing the mouse is pointing at, and call this with the widget (this pointer is used to remove the tooltip if
the widget is deleted or hidden, and to locate the tooltip), the rectangle surrounding the area, relative to the top-left
corner of the widget (used to calculate where to put the tooltip), and the text of the tooltip (which must be a pointer to static data as it is not copied).

12.151.2.10  font() [1/2]

static Fl_Font Fl_Tooltip::font ( ) [inline], [static]
Gets the typeface for the tooltip text.

12.151.2.11  font() [2/2]

static void Fl_Tooltip::font (Fl_Font i) [inline], [static]
Sets the typeface for the tooltip text.

12.151.2.12  hidedelay() [1/2]

static float Fl_Tooltip::hidedelay ( ) [inline], [static]
Gets the time until an open tooltip hides again.
The default delay is 12.0 seconds.

12.151.2.13  hidedelay() [2/2]

static void Fl_Tooltip::hidedelay (float f) [inline], [static]
Sets the time until an open tooltip hides again.
The default delay is 12.0 seconds.

12.151.2.14  hoverdelay() [1/2]

static float Fl_Tooltip::hoverdelay ( ) [inline], [static]
Gets the tooltip hover delay, the delay between tooltips.
The default delay is 0.2 seconds.

12.151.2.15  hoverdelay() [2/2]

static void Fl_Tooltip::hoverdelay (float f) [inline], [static]
Sets the tooltip hover delay, the delay between tooltips.
The default delay is 0.2 seconds.

12.151.2.16  margin_height() [1/2]

static int Fl_Tooltip::margin_height ( ) [inline], [static]
Gets the amount of extra space above and below the tooltip's text.
Default is 3.

12.151.2.17  margin_height() [2/2]

static void Fl_Tooltip::margin_height (int v) [inline], [static]
Sets the amount of extra space above and below the tooltip's text.
Default is 3.
12.151.2.18 margin_width() [1/2]

static int Fl_Tooltip::margin_width ( ) [inline], [static]

Gets the amount of extra space left/right of the tooltip's text.
Default is 3.

12.151.2.19 margin_width() [2/2]

static void Fl_Tooltip::margin_width ( int v ) [inline], [static]

Sets the amount of extra space left/right of the tooltip's text.
Default is 3.

12.151.2.20 size() [1/2]

static Fl_Fontsize Fl_Tooltip::size ( ) [inline], [static]

Gets the size of the tooltip text.

12.151.2.21 size() [2/2]

static void Fl_Tooltip::size ( Fl_Fontsize s ) [inline], [static]

Sets the size of the tooltip text.

12.151.2.22 textcolor() [1/2]

static Fl_Color Fl_Tooltip::textcolor ( ) [inline], [static]

Gets the color of the text in the tooltip.
The default is black.

12.151.2.23 textcolor() [2/2]

static void Fl_Tooltip::textcolor ( Fl_Color c ) [inline], [static]

Sets the color of the text in the tooltip.
The default is black.

12.151.2.24 wrap_width() [1/2]

static int Fl_Tooltip::wrap_width ( ) [inline], [static]

Gets the maximum width for tooltip's text before it word wraps.
Default is 400.

12.151.2.25 wrap_width() [2/2]

static void Fl_Tooltip::wrap_width ( int v ) [inline], [static]

Sets the maximum width for tooltip's text before it word wraps.
Default is 400.

The documentation for this class was generated from the following files:

- Fl_Tooltip.H
- Fl.cxx
- Fl_Tooltip.cxx
12.152  **Fl_Tree Class Reference**

Tree widget.

```cpp
#include <Fl_Tree.H>
```

Inheritance diagram for Fl_Tree:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Tree
```

**Public Member Functions**

- `Fl_Tree_Item * add (const char *path, Fl_Tree_Item *newitem=0)`
  Adds a new item, given a menu style 'path'.
- `Fl_Tree_Item * add (Fl_Tree_Item *parent_item, const char *name)`
  Add a new child item labeled 'name' to the specified 'parent_item'.
- `void calc_dimensions ()`
  Recalculate widget dimensions and scrollbar visibility, normally managed automatically.
- `void calc_tree ()`
  Recalculates the tree’s sizes and scrollbar visibility, normally managed automatically.
- `Fl_Tree_Item * callback_item ()`
  Gets the item that caused the callback.
- `void callback_item (Fl_Tree_Item *item)`
  Sets the item that was changed for this callback.
- `Fl_Tree_Reason callback_reason () const`
  Gets the reason for this callback.
- `void callback_reason (Fl_Tree_Reason reason)`
  Sets the reason for this callback.
- `void clear ()`
  Clear the entire tree’s children, including the root.
- `void clear_children (Fl_Tree_Item *item)`
  Clear all the children for 'item'.
- `int close (const char *path, int docallback=1)`
  Closes the item specified by 'path'.
- `int close (Fl_Tree_Item *item, int docallback=1)`
  Closes the specified 'item'.
- `Fl_Image * closeicon () const`
  Returns the icon to be used as the ‘close’ icon.
- `void closeicon (Fl_Image *val)`
  Sets the icon to be used as the ‘close’ icon.
- `Fl_Color connectorcolor () const`
  Get the connector color used for tree connection lines.
- `void connectorcolor (Fl_Color val)`
  Set the connector color used for tree connection lines.
- `Fl_Tree_Connector connectorstyle () const`
  Returns the line drawing style for inter-connecting items.
- `void connectorstyle (Fl_Tree_Connector val)`
  Sets the line drawing style for inter-connecting items.
• int connectorwidth () const
  
  Gets the width of the horizontal connection lines (in pixels) that appear to the left of each tree item's label.

• void connectorwidth (int val)
  
  Sets the width of the horizontal connection lines (in pixels) that appear to the left of each tree item's label.

• int deselect (const char *path, int docallback=1)
  
  Deselect an item specified by 'path'.

• int deselect (Fl_Tree_Item *item, int docallback=1)
  
  Deselect the specified item.

• int deselect_all (Fl_Tree_Item *item=0, int docallback=1)
  
  Deselect 'item' and all its children.

• void display (Fl_Tree_Item *item)
  
  Displays 'item', scrolling the tree as necessary.

• int displayed (Fl_Tree_Item *item)
  
  See if 'item' is currently displayed on-screen (visible within the widget).

• int extend_selection (Fl_Tree_Item *from, Fl_Tree_Item *to, int val=1, bool visible=false)
  
  Extend a selection between 'from' and 'to' depending on 'visible'.

• int extend_selection_dir (Fl_Tree_Item *from, Fl_Tree_Item *to, int dir, int val, bool visible)
  
  Extend the selection between and including 'from' and 'to' depending on direction 'dir', 'val', and 'visible'.

• Fl_Tree_Item *find_clicked (int yonly=0)
  
  Non-const version of Fl_Tree::find_clicked(int yonly) const.

• const Fl_Tree_Item *find_clicked (int yonly=0) const
  
  Find the item that was last clicked on.

• Fl_Tree_Item *find_item (const char *path)
  
  Non-const version of Fl_Tree::find_item(const char *path) const.

• const Fl_Tree_Item *find_item (const char *path) const
  
  Find the item, given a menu style path, e.g.

• Fl_Tree_Item *first ()
  
  Returns the first item in the tree, or 0 if none.

• Fl_Tree_Item *first_selected_item ()
  
  Returns the first selected item in the tree.

• Fl_Tree_Item *first_visible ()
  
  Returns the first open(), visible item in the tree, or 0 if none.

• Fl_Tree_Item *first_visible_item ()
  
  Returns the first open(), visible item in the tree, or 0 if none.

• Fl_Tree (int X, int Y, int W, int H, const char *L=0)
  
  Constructor.

• Fl_Tree_Item *get_item_focus () const
  
  Get the item that currently has keyboard focus.

• int get_selected_items (Fl_Tree_Item_Array &items)
  
  Returns the currently selected items as an array of 'ret_items'.

• int handle (int e) FL_OVERRIDE
  
  Standard FLTK event handler for this widget.

• int hposition () const
  
  Returns the horizontal scroll position as a pixel offset.

• void hposition (int pos)
  
  Sets the horizontal scroll offset to position 'pos'.

• Fl_Tree_Item *insert (Fl_Tree_Item *item, const char *name, int pos)
  
  Insert a new item 'name' into 'item's children at position 'pos'.

• Fl_Tree_Item *insert_above (Fl_Tree_Item *above, const char *name)
  
  Inserts a new item 'name' above the specified Fl_Tree_Item 'above'.

Generated by Doxygen
• int is_close (const char *path) const
  See if item specified by 'path' is closed.
• int is_close (Fl_Tree_Item *item) const
  See if the specified 'item' is closed.
• int is_hscroll_visible () const
  See if the horizontal scrollbar is currently visible.
• int is_open (const char *path) const
  See if item specified by 'path' is open.
• int is_open (Fl_Tree_Item *item) const
  See if 'item' is open.
• int is_scrollbar (Fl_Widget *w)
  See if widget 'w' is one of the Fl_Tree widget's scrollbars.
• int is_selected (const char *path)
  See if item specified by 'path' is selected.
• int is_selected (Fl_Tree_Item *item) const
  See if the specified 'item' is selected.
• int is_vscroll_visible () const
  See if the vertical scrollbar is currently visible.
• Fl_Tree_Item *item_clicked ()
  Return the item that was last clicked.
• Fl_Tree_Item_Draw_Mode item_draw_mode () const
  Get the 'item draw mode' used for the tree.
• void item_draw_mode (Fl_Tree_Item_Draw_Mode mode)
  Set the 'item draw mode' used for the tree to 'mode'.
• void item_draw_mode (int mode)
  Set the 'item draw mode' used for the tree to integer 'mode'.
• void item_labelbgcolor (Fl_Color val)
  Set the default label background color used for creating new items.
• Fl_Color item_labelbgcolor (void) const
  Get the default label background color used for creating new items.
• void item_labelfgcolor (Fl_Color val)
  Set the default label foreground color used for creating new items.
• Fl_Color item_labelfgcolor (void) const
  Get the default label foreground color used for creating new items.
• Fl_Font item_labelfont () const
  Get the default font face used for creating new items.
• void item_labelfont (Fl_Font val)
  Set the default font face used for creating new items.
• Fl_Fontsize item_labelsize () const
  Get the default label fontsize used for creating new items.
• void item_labelsize (Fl_Fontsize val)
  Set the default label font size used for creating new items.
• int item_pathname (char *pathname, int pathnamelen, const Fl_Tree_Item *item) const
  Return 'pathname' of size 'pathnamelen' for the specified 'item'.
• Fl_Tree_Item_Reselect_Mode item_reselect_mode () const
  Returns the current item re/selection mode.
• void item_reselect_mode (Fl_Tree_Item_Reselect_Mode mode)
  Sets the item re/selection mode.
• int labelmarginleft () const
  Get the amount of white space (in pixels) that should appear to the left of the label text.
• void labelmarginleft (int val)
Set the amount of white space (in pixels) that should appear to the left of the label text.

- **Fl_Tree_Item** * last ()
  Returns the last item in the tree.
- **Fl_Tree_Item** * last_selected_item ()
  Returns the last selected item in the tree.
- **Fl_Tree_Item** * last_visible ()
  Returns the last open(), visible item in the tree.
- **Fl_Tree_Item** * last_visible_item ()
  Returns the last open(), visible item in the tree.

• **int** linespacing () const
  Get the amount of white space (in pixels) that should appear between items in the tree.

• **void** linespacing (int val)
  Sets the amount of white space (in pixels) that should appear between items in the tree.

• **void** load (class Fl_Preferences &)
  Load FLTK preferences.

• **int** marginbottom () const
  Get the amount of white space (in pixels) that should appear below the last visible item when the vertical scroller is scrolled to the bottom.

• **void** marginbottom (int val)
  Sets the amount of white space (in pixels) that should appear below the last visible item when the vertical scroller is scrolled to the bottom.

• **int** marginleft () const
  Get the amount of white space (in pixels) that should appear between the widget’s left border and the tree’s contents.

• **void** marginleft (int val)
  Set the amount of white space (in pixels) that should appear between the widget’s left border and the left side of the tree’s contents.

• **int** margintop () const
  Get the amount of white space (in pixels) that should appear between the widget’s top border and the top of the tree’s contents.

• **void** margintop (int val)
  Sets the amount of white space (in pixels) that should appear between the widget’s top border and the top of the tree’s contents.

- **Fl_Tree_Item** * next (Fl_Tree_Item *item=0)
  Return the next item after 'item', or 0 if no more items.

- **Fl_Tree_Item** * next_item (Fl_Tree_Item *item, int dir=FL_Down, bool visible=false)
  Returns next item after 'item' in direction 'dir' depending on 'visible'.

- **Fl_Tree_Item** * next_selected_item (Fl_Tree_Item *item=0, int dir=FL_Down)
  Returns the next selected item above or below 'item', depending on 'dir'.

- **Fl_Tree_Item** * next_visible_item (Fl_Tree_Item *start, int dir)
  Returns next open(), visible item above (dir==FL_Up) or below (dir==FL_Down) the specified 'item', or 0 if no more items.

• **int** open (const char *path, int docallback=1)
  Opens the item specified by 'path'.

• **int** open (Fl_Tree_Item *item, int docallback=1)
  Open the specified 'item'.

• **void** open_toggle (Fl_Tree_Item *item, int docallback=1)
  Toggle the open state of 'item'.

• **int** openchild_marginbottom () const
  Get the amount of white space (in pixels) that should appear below an open child tree's contents.

• **void** openchild_marginbottom (int val)
  Set the amount of white space (in pixels) that should appear below an open child tree's contents.

• **Fl_Image** * openicon () const
Returns the icon to be used as the 'open' icon.

- void openicon (Fl_Image *val)
  
  Sets the icon to be used as the 'open' icon.

- const Fl_Tree_Prefs & prefs () const

- Fl_Tree_Item * prev (Fl_Tree_Item *item=0)
  
  Return the previous item before 'item', or 0 if no more items.

- void recalc_tree ()
  
  Schedule tree to recalc the entire tree size.

- int remove (Fl_Tree_Item *item)
  
  Remove the specified 'item' from the tree.

- void resize (int, int, int, int) FL_OVERRIDE
  
  Resizes the Fl_Group widget and all of its children.

- Fl_Tree_Item * root ()
  
  Returns the root item.

- void root (Fl_Tree_Item *newitem)
  
  Sets the root item to 'newitem'.

- void root_label (const char *new_label)
  
  Set the label for the root item to 'new_label'.

- int scrollbar_size () const
  
  Gets the default size of scrollbars' troughs for this widget in pixels.

- void scrollbar_size (int size)
  
  Sets the pixel size of the scrollbars' troughs to 'size' for this widget, in pixels.

- int select (const char *path, int docallback=1)
  
  Select the item specified by 'path'.

- int select (Fl_Tree_Item *item, int docallback=1)
  
  Select the specified 'item'.

- int select_all (Fl_Tree_Item *item=0, int docallback=1)
  
  Select 'item' and all its children.

- int select_only (Fl_Tree_Item *selitem, int docallback=1)
  
  Select only the specified item, deselecting all others that might be selected.

- void select_toggle (Fl_Tree_Item *item, int docallback=1)
  
  Toggle the select state of the specified 'item'.

- Fl_Boxtype selectbox () const
  
  Sets the style of box used to draw selected items.

- void selectbox (Fl_Boxtype val)
  
  Gets the style of box used to draw selected items.

- Fl_Tree_Select selectmode () const
  
  Gets the tree's current selection mode.

- void selectmode (Fl_Tree_Select val)
  
  Sets the tree's selection mode.

- void set_item_focus (Fl_Tree_Item *item)
  
  Set the item that currently should have keyboard focus.

- void show_item (Fl_Tree_Item *item)
  
  Adjust the vertical scrollbar to show 'item' at the top of the display IF it is currently off-screen (for instance show_item_top()).

- void show_item (Fl_Tree_Item *item, int yoff)
  
  Adjust the vertical scrollbar so that 'item' is visible 'yoff' pixels from the top of the Fl_Tree widget's display.

- void show_item_bottom (Fl_Tree_Item *item)
  
  Adjust the vertical scrollbar so that 'item' is at the bottom of the display.

- void show_item_middle (Fl_Tree_Item *item)
  
  Adjust the vertical scrollbar so that 'item' is in the middle of the display.
• void show_item_top (Fl_Tree_Item *item)
  Adjust the vertical scrollbar so that 'item' is at the top of the display.

• void show_self ()
  Print the tree as 'ascii art' to stdout.

• int showcollapse () const
  Returns 1 if the collapse icon is enabled, 0 if not.

• void showcollapse (int val)
  Set if we should show the collapse icon or not.

• int showroot () const
  Returns 1 if the root item is to be shown, or 0 if not.

• void showroot (int val)
  Set if the root item should be shown or not.

• Fl_Tree_Sort sortorder () const
  Set the default sort order used when items are added to the tree.

• void sortorder (Fl_Tree_Sort val)
  Gets the sort order used to add items to the tree.

• Fl_Image * usericon () const
  Returns the Fl_Image being used as the default user icon for all newly created items.

• void usericon (Fl_Image *val)
  Sets the Fl_Image to be used as the default user icon for all newly created items.

• int usericonmarginleft () const
  Get the amount of white space (in pixels) that should appear to the left of the usericon.

• void usericonmarginleft (int val)
  Set the amount of white space (in pixels) that should appear to the left of the usericon.

• int vposition () const
  Returns the vertical scroll position as a pixel offset.

• void vposition (int pos)
  Sets the vertical scroll offset to position 'pos'.

• int widgetmarginleft () const
  Get the amount of white space (in pixels) that should appear to the left of the child Fltk widget (if any).

• void widgetmarginleft (int val)
  Set the amount of white space (in pixels) that should appear to the left of the child Fltk widget (if any).

• ~Fl_Tree ()
  Destructor.

Public Member Functions inherited from Fl_Group

• Fl_Widget *& _ddfdesign_kludge ()
  This is for forms compatibility only.

• void add (Fl_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.

• void add (Fl_Widget &o)
  See void Fl_Group::add(Fl_Widget &w)

• void add_resizable (Fl_Widget &o)
  Adds a widget to the group and makes it the resizable widget.

• Fl_Widget * const * array () const
  Returns a pointer to the array of children.

• Fl_Group const * as_group () const FL_OVERRIDE

• Fl_Group * as_group () FL_OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• void begin ()
  Destructor.
Sets the current group so you can build the widget tree by just constructing the widgets.

- **Fl_Widget** + child (int n) const
  Returns array\[n\].
- int children () const
  Returns how many child widgets the group has.
- void clear ()
  Deletes all child widgets from memory recursively.
- unsigned int clip_children ()
  Returns the current clipping mode.
- void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.
- void end ()
  Exactly the same as current(this->parent()).
- int find (const Fl_Widget &o) const
  See int Fl_Group::find(const Fl_Widget *w) const.
- int find (const Fl_Widget *) const
  Searches the child array for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *0)
  Creates a new Fl_Group widget using the given position, size, and label string.
- void focus (Fl_Widget &W)
- void forms_end ()
  This is for forms compatibility only.
- void init_sizes ()
  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)
  This does insert(w, find(before)).
- void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)
  Removes the widget o from the group.
- void remove (int index)
  Removes the widget at index from the group but does not delete it.
- Fl_Widget *resizable () const
  Returns the group’s resizable widget.
- void resizable (Fl_Widget &o)
  Sets the group’s resizable widget.
- void resizable (Fl_Widget *)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()
  The destructor also deletes all the children.
Public Member Functions inherited from Fl_Widget

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ as_gl_window () const
- virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ as_window () const
- void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const
  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
- Fl_Callback_p callback () const
  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗ cb, void ∗ p)
  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗ cb)
  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗ cb, long p=0)
  Sets the current callback function for the widget.
- unsigned int changed () const
  Checks if the widget value changed since the last callback.
- void clear_active ()
Marks the widget as inactive without sending events or changing focus.

- `void clear_changed ()`
  Marks the value of the widget as unchanged.

- `void clear_damage (uchar c=0)`
  Clears or sets the damage flags.

- `void clear_output ()`
  Sets a widget to accept input.

- `void clear_visible ()`
  Hides the widget.

- `void clear_visible_focus ()`
  Disables keyboard focus navigation with this widget.

- `Fl_Color color () const`
  Gets the background color of the widget.

- `void color (Fl_Color bg)`
  Sets the background color of the widget.

- `void color (Fl_Color bg, Fl_Color sel)`
  Sets the background and selection color of the widget.

- `Fl_Color color2 () const`
  For back compatibility only.

- `void color2 (unsigned a)`
  For back compatibility only.

- `int contains (const Fl_Widget *w) const`
  Checks if w is a child of this widget.

- `void copy_label (const char *new_label)`
  Sets the current label.

- `void copy_tooltip (const char *text)`
  Sets the current tooltip text.

- `uchar damage () const`
  Returns non-zero if draw() needs to be called.

- `void damage (uchar c)`
  Sets the damage bits for the widget.

- `void damage (uchar c, int x, int y, int w, int h)`
  Sets the damage bits for an area inside the widget.

- `int damage_resize (int, int, int, int)`
  Internal use only.

- `void deactivate ()`
  Deactivates the widget.

- `Fl_Image *deimage ()`
  Gets the image that is used as part of the widget label when in the inactive state.

- `const Fl_Image *deimage () const`
  Gets the image that is used as part of the widget label when in the inactive state.

- `void deimage (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the inactive state.

- `void deimage (Fl_Image *img)`
  Sets the image to use as part of the widget label when in the inactive state.

- `int deimage_bound () const`
  Returns whether the inactive image is managed by the widget.

- `void do_callback (Fl_Callback_Reasnick reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with default arguments.

- `void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reasnick reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.
• void *do_callback (Fl_Widget ∗widget, void ∗arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
  Gets the widget height.
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image ∗image ()
  Gets the image that is used as part of the widget label when in the active state.
• const Fl_Image ∗image () const
  Gets the image that is used as part of the widget label when in the active state.
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.
• void image (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the active state.
• int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget ∗wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char ∗label () const
  Gets the current label text.
• void label (const char ∗text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char ∗b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labeltype () const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
Returns if a widget is used for output only.

- **Fl_Group** * parent () const
  
  Returns a pointer to the parent widget.

- void parent (Fl_Group * p)
  
  Internal use only - “for hacks only”.

- void position (int X, int Y)
  
  Repositions the window or widget.

- void redraw ()
  
  Schedules the drawing of the widget.

- void redraw_label ()
  
  Schedules the drawing of the label.

- **Fl_Color** selection_color () const
  
  Gets the selection color.

- void selection_color (Fl_Color a)
  
  Sets the selection color.

- void set_active ()
  
  Marks the widget as active without sending events or changing focus.

- void set_changed ()
  
  Marks the value of the widget as changed.

- void set_output ()
  
  Sets a widget to output only.

- void set_visible ()
  
  Makes the widget visible.

- void set_visible_focus ()
  
  Enables keyboard focus navigation with this widget.

- int shortcut_label () const
  
  Returns whether the widget's label uses '&' to indicate shortcuts.

- void shortcut_label (int value)
  
  Sets whether the widget's label uses '&' to indicate shortcuts.

- virtual void show ()
  
  Makes a widget visible.

- void size (int W, int H)
  
  Changes the size of the widget.

- int take_focus ()
  
  Gives the widget the keyboard focus.

- unsigned int takesevents () const
  
  Returns if the widget is able to take events.

- int testShortcut ()
  
  Returns true if the widget's label contains the entered 'x' shortcut.

- const char * tooltip () const
  
  Gets the current tooltip text.

- void tooltip (const char * text)
  
  Sets the current tooltip text.

- **Fl_Window** * top_window () const
  
  Returns a pointer to the top-level window for the widget.

- **Fl_Window** * top_window_offset (int & xoff, int & yoff) const
  
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const
  
  Gets the widget type.

- void type (uchar t)
  
  Sets the widget type.
• int **use_accents_menu** ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void *user_data() const
  Gets the user data for this widget.
• void **user_data** (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void **user_data** (void *v)
  Sets the user data for this widget.
• unsigned int **visible()** const
  Returns whether a widget is visible.
• unsigned int **visible_focus()** const
  Checks whether this widget has a visible focus.
• void **visible_focus**(int v)
  Modifies keyboard focus navigation.
• int **visible_r()** const
  Returns whether a widget and all its parents are visible.
• int **w()** const
  Gets the widget width.
• Fl_When **when()** const
  Returns the conditions under which the callback is called.
• void **when**(uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * **window()** const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int **x()** const
  Gets the widget position in its window.
• int **y()** const
  Gets the widget position in its window.
• virtual ~**Fl_Widget** ()
  Destroys the widget.

**Protected Member Functions**

• void **do_callback_for_item** (Fl_Tree_Item *item, Fl_Tree_Reason reason)
  Do the callback for the specified 'item' using 'reason', setting the callback_item() and callback_reason().
• void **draw** () FL_OVERRIDE
  Standard FLTK draw() method, handles drawing the tree widget.
• void **item_clicked**(Fl_Tree_Item *val)
  Set the item that was last clicked.

**Protected Member Functions inherited from Fl_Group**

• Fl_Rect * **bounds()**
  Returns the internal array of widget sizes and positions.
• void **draw_child**(Fl_Widget &widget) const
  Forces a child to redraw.
• void **draw_children()**
  Draws all children of the group.
• void **draw_outside_label**(const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.
• virtual int **on_insert**(Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.
virtual int on_move (int, int)  
Allow derived groups to act when a widget is moved within the group.

virtual void on_remove (int)  
Allow derived groups to act when a child widget is removed from the group.

int * sizes ()  
Returns the internal array of widget sizes and positions.

void update_child (Fl_Widget &widget) const  
Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

void clear_flag (unsigned int c)  
Clears a flag in the flags mask.

void draw_backdrop () const  
If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

void draw_box () const  
Draws the widget box according its box style.

void draw_box (Fl_Boxtype t, Fl_Color c) const  
Draws a box of type t, of color c at the widget's position and size.

void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const  
Draws a box of type t, of color c at the position X,Y and size W,H.

void draw_focus () const  
Draws a focus rectangle around the widget.

void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const  
Draws a focus rectangle around the widget.

void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const  
Draws a focus box for the widget at the given position and size.

void draw_label () const  
Draws the widget's label at the defined label position.

void draw_label (int, int, int, int) const  
Draws the label in an arbitrary bounding box.

void h (int v)  
Internal use only.

void set_flag (unsigned int c)  
Sets a flag in the flags mask.

void w (int v)  
Internal use only.

void x (int v)  
Internal use only.

void y (int v)  
Internal use only.
Protected Attributes

- **Fl_Scrollbar _hscroll**
  Horizontal scrollbar.
- int _tih
  Tree widget inner xywh dimension: inside borders + scrollbars.
- int _tiw
- int _tix
- int _tiy
- int _toh
  Tree widget outer xywh dimension: outside scrollbars, inside widget border.
- int _tow
- int _tox
- int _toy
- int _tree_h
  the calculated height of the entire tree hierarchy. See calc_tree()
- int _tree_w
  the calculated width of the entire tree hierarchy. See calc_tree()
- **Fl_Scrollbar _vscroll**
  Vertical scrollbar.

Friends

- class Fl_Tree_Item

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group *current ()
  Returns the currently active group.
- static void current (Fl_Group *g)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the 'x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text contains the entered 'x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16 , COPYED_TOOLSHIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCECTS_MENU
  = 1<<19 ,
  NEEDS_KEYBBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22
}
AUTO_DELETE_USER_DATA = 1<<23,
MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }
flags possible values enumeration.

12.152.1 Detailed Description

Tree widget.

Figure 12.62 Fl_Tree example program

Fl_Tree  // Top level widget
|--- Fl_Tree_Item  // Items in the tree
|--- Fl_Tree_Prefs  // Preferences for the tree
|   |--- Fl_Tree_Connector (enum) // Connection modes
|   |--- Fl_Tree_Select (enum) // Selection modes
|--- Fl_Tree_Sort (enum) // Sort behavior

Similar to Fl_Browser, Fl_Tree is a browser of Fl_Tree_Item’s arranged in a parented hierarchy, or ‘tree’. Subtrees can be expanded or closed. Items can be added, deleted, inserted, sorted and re-ordered.
The tree items may also contain other FLTK widgets, like buttons, input fields, or even "custom" widgets.
The callback() is invoked depending on the value of when():

- FL_WHEN_RELEASE – callback invoked when left mouse button is released on an item
- FL_WHEN_CHANGED – callback invoked when left mouse changes selection state

The simple way to define a tree:

```c
#include <FL/Fl_Tree.H>
[..]
Fl_Tree tree(X,Y,W,H);
tree.begin();
tree.add("Flintstones/Fred");
tree.add("Flintstones/Wilma");
tree.add("Flintstones/Pebbles");
tree.add("Simpsons/Homer");
tree.add("Simpsons/Marge");
tree.add("Simpsons/Bart");
tree.add("Simpsons/Lisa");
tree.end();
```

FEATURES

- Items can be added with add(), removed with remove(), completely cleared with clear(), inserted with insert() and insert_above(), selected/deselected with select() and deselect(), open/closed with open() and close(), positioned on the screen with show_item_top(), show_item_middle() and show_item_bottom(),
- item children can be swapped around with Fl_Tree_Item::swap_children(),
- items can be moved around with Fl_Tree_Item::move(),

Generated by Doxygen
an item's children can be walked with Fl_Tree_Item::first() and Fl_Tree_Item::next(), an item's children can be
indexed directly with Fl_Tree_Item::child() and Fl_Tree_Item::children(),
items can be moved from one subtree to another with Fl_Tree_Item::deparent() and Fl_Tree_Item::reparent(),
sorting can be controlled when items are add(ED via sortorder()).
You can walk the entire tree with first() and next().
You can walk visible items with first_visible_item() and next_visible_item().
You can walk selected items with first_selected_item() and next_selected_item().
Items can be found by their pathname using find_item(const char *), and an item's pathname can be found
with item_pathname().
The selected items' colors are controlled by selection_color() (inherited from Fl_Widget).
A hook is provided to allow you to redefine how item's labels are drawn via Fl_Tree::item_draw_callback().
Items can be interactively dragged using FL_TREE_SELECT_SINGLE_DRAGGABLE.

SELECTION OF ITEMS
The tree can have different selection behaviors controlled by selectmode(). The background color used for
selected items is the Fl_Tree::selection_color(). The foreground color for selected items is controlled internally
with fl_contrast().

CHILD WIDGETS
FLTK widgets (including custom widgets) can be assigned to tree items via Fl_Tree_Item::widget().

When an Fl_Tree_Item::widget() is defined, the default behavior is for the widget() to be shown in place of the
item's label (if it has one). Only the widget()'s width will be used; the widget()'s x() and y() position will be
managed by the tree, and the h() will track the item's height. This default behavior can be altered (ABI 1.3.1):
Setting Fl_Tree::item_draw_mode()'s FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET flag causes the label
+ widget to be displayed together in that order, and adding the FL_TREE_ITEM_HEIGHT_FROM_WIDGET
flag causes widget's height to define the widget()'s height.

ICONS
The tree's open/close icons can be redefined with Fl_Tree::openicon(), Fl_Tree::closeicon(). User icons can
either be changed globally with Fl_Tree::usericon(), or on a per-item basis with Fl_Tree_Item::usericon().

Various default preferences can be globally manipulated via Fl_Tree_Prefs, including colors, margins, icons,
connection lines, etc.

FONTS AND COLORS
When adding new items to the tree, the new items get the defaults for fonts and colors from:

- Fl_Tree::item_labelfont() – The default item label font (default: FL_HELVETICA)
- Fl_Tree::item_labelsize() – The default item label size (default: FL_NORMAL_SIZE)
- Fl_Tree::item_labelfgcolor() – The default item label foreground color (default: FL_FOREGROUND_COLOR)
- Fl_Tree::item_labelbgcolor() – The default item label background color (default: 0xffffffff, which tree uses
  as 'transparent')

Each item (Fl_Tree_Item) inherits a copy of these font/color attributes when created, and each item has its
own methods to let the app change these values on a per-item basis using methods of the same name:
• `Fl_Tree_Item::labelfont()` – The item's label font (default: FL_HELVETICA)
• `Fl_Tree_Item::labelsize()` – The item's label size (default: FL_NORMAL_SIZE)
• `Fl_Tree_Item::labelfgcolor()` – The item's label foreground color (default: FL_FOREGROUND_COLOR)
• `Fl_Tree_Item::labelbgcolor()` – The item's label background color (default: 0xffffffff, which uses the tree's own bg color)

CALLBACKS

The tree's `callback()` will be invoked when items change state or are open/closed. `when()` controls when mouse/keyboard events invoke the callback. `callback_item()` and `callback_reason()` can be used to determine the cause of the callback. e.g.

```c
void MyTreeCallback(Fl_Widget *w, void *data) {
    Fl_Tree *tree = (Fl_Tree*)w;
    Fl_Tree_Item *item = (Fl_Tree_Item*)tree->callback_item(); // get selected item

    switch (tree->callback_reason()) {
        case FL_TREE_REASON_SELECTED: [..]
        case FL_TREE_REASON_DESELECTED: [..]
        case FL_TREE_REASON_RESELECTED: [..]
        case FL_TREE_REASON_OPENED: [..]
        case FL_TREE_REASON_CLOSED: [..]
    }
}
```

SIMPLE EXAMPLES

To find all the selected items:

```c
for ( Fl_Tree_Item *i=first_selected_item(); i; i=next_selected_item(i) )
    printf("Item %s is selected
", i->label());
```

To get an item's full menu pathname, use `Fl_Tree::item_pathname()`, e.g.

```c
char pathname[256] = "???";
tree->item_pathname(pathname, sizeof(pathname), item); // eg. "Parent/Child/Item"
```

To walk all the items of the tree from top to bottom:

```c
// Walk all the items in the tree, and print their labels
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->next(item) ) {
    printf("Item: %s
", item->label());
}
```

To recursively walk all the children of a particular item, define a function that uses recursion:

```c
// Find all of the item's children and print an indented report of their labels
void my_print_all_children(Fl_Tree_Item *item, int indent=0) {
    for ( int t=0; t<item->children(); t++ ) {
        printf("%*s Item: %s
", indent, "", item->child(t)->label());
        my_print_all_children(item->child(t), indent+4); // recurse
    }
}
```
To change the default label font and color when creating new items:

```cpp
    tree = new Fl_Tree(..);
    tree->item_labelfont(FL_COURIER); // Use Courier font for all new items
    tree->item_labelfgcolor(FL_RED); // Use red color for labels of all new items
    // Now create the items in the tree using the above defaults.
    tree->add("Aaa");
    tree->add("Bbb");
```

To change the font and color of all existing items in the tree:

```cpp
    // Change the font and color of all items currently in the tree
    for ( Fl_Tree_Item *item = tree->first(); item; item = tree->next(item) ) {
        item->labelfont(FL_COURIER);
        item->labelcolor(FL_RED);
    }
```

**DISPLAY DESCRIPTION**

The following image shows the tree's various visual elements and the methods that control them:

![Figure 12.63 Fl_Tree elements](image)

The following shows the protected dimension variables 'tree inner' (tix..) and 'tree outer' (tox..):

![Figure 12.64 Fl_Tree inner/outer dimensions](image)
KEYBOARD BINDINGS

The following table lists keyboard bindings for navigating the tree:

<table>
<thead>
<tr>
<th>Keyboard</th>
<th>FL_TREE_SELECT&lt;-_MULTI</th>
<th>FL_TREE_SELECT&lt;-_SINGLE</th>
<th>FL_TREE_SELECT&lt;-_NONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl-A (Linux/Windows)</td>
<td>Select all items</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Command-A (Mac)</td>
<td>Select all items</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Space</td>
<td>Selects item</td>
<td>Selects item</td>
<td>N/A</td>
</tr>
<tr>
<td>Ctrl-Space</td>
<td>Toggle item</td>
<td>Toggle item</td>
<td>N/A</td>
</tr>
<tr>
<td>Shift-Space</td>
<td>Extends selection</td>
<td>Selects item</td>
<td>N/A</td>
</tr>
<tr>
<td>Enter</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
</tr>
<tr>
<td>Ctrl-Enter</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
</tr>
<tr>
<td>Shift-Enter</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
</tr>
<tr>
<td>Right / Left</td>
<td>Open/Closed item</td>
<td>Open/Closed item</td>
<td>Open/Closed item</td>
</tr>
<tr>
<td>Up / Down</td>
<td>Move focus box up/down</td>
<td>Move focus box up/down</td>
<td>N/A</td>
</tr>
<tr>
<td>Shift-Up / Shift-Down</td>
<td>Extend selection up/down</td>
<td>Move focus up/down</td>
<td>N/A</td>
</tr>
<tr>
<td>Home / End</td>
<td>Move to top/bottom of tree</td>
<td>Move to top/bottom of tree</td>
<td>Move to top/bottom of tree</td>
</tr>
<tr>
<td>PageUp / PageDown</td>
<td>Page up/down</td>
<td>Page up/down</td>
<td>Page up/down</td>
</tr>
</tbody>
</table>

12.152.2 Member Function Documentation

12.152.2.1 add() [1/2]

\[ Fl_Tree_Item * Fl_Tree::add ( \]
\[ \text{const char * path,} \]
\[ \text{Fl_Tree_Item * item = 0 )} \]

Adds a new item, given a menu style 'path'. Any parent nodes that don't already exist are created automatically. Adds the item based on the value of sortorder(). If 'item' is NULL, a new item is created. To specify items or submenus that contain slashes ('/' or '\') use an escape character to protect them, e.g.

```
: tree->add("/Holidays/Photos/12\25\2010"); // Adds item "12/25/2010"
: tree->add("/Pathnames/c:\\Program Files\\MyApp");  // Adds item "c:\Program Files\MyApp"
```

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>The path to the item, e.g. &quot;Flintstone/Fred&quot;.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>item</td>
<td>The new item to be added. If NULL, a new item is created with a name that is the last element in 'path'.</td>
</tr>
</tbody>
</table>

Returns

The new item added, or 0 on error.
**12.152.2.2 add()** [2/2]

```c
Fl_Tree_Item * Fl_Tree::add (  
    Fl_Tree_Item * parent_item,  
    const char * name )
```

Add a new child item labeled 'name' to the specified 'parent_item'.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>parent_item</th>
<th>The parent item the new child item will be added to. Must not be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>The label for the new item</td>
</tr>
</tbody>
</table>

**Returns**

The new item added.

**Version**

1.3.3

---

**12.152.2.3 calc_dimensions()**

```c
void Fl_Tree::calc_dimensions ( )
```

Recalculate widget dimensions and scrollbar visibility, normally managed automatically.

Low overhead way to update the tree widget's outer/inner dimensions and re-determine scrollbar visibility based on these changes without recalculating the entire size of the tree data.

Assumes that either the tree's size in _tree_w/_tree_h are correct so that scrollbar visibility can be calculated easily, or are both zero indicating scrollbar visibility can't be calculated yet.

This method is called when the widget is resized or if the scrollbar's sizes are changed (affects tree widget's inner dimensions tix/y/w/h), and also used by calc_tree().

**Version**

1.3.3 ABI feature

---

**12.152.2.4 calc_tree()**

```c
void Fl_Tree::calc_tree ( )
```

Recalculates the tree's sizes and scrollbar visibility, normally managed automatically.

On return:

- _tree_w will be the overall pixel width of the entire viewable tree
- _tree_h will be the overall pixel height
- scrollbar visibility and pan sizes are updated
- internal _tix/_tiy/_tiw/_tih dimensions are updated

_tree_w/_tree_h include the tree's margins (e.g. marginleft()), whether items are open or closed, label contents and font sizes, etc.

The tree hierarchy's size is managed separately from the widget's size as an optimization; this way resize() on the widget doesn't involve recalculating the tree's hierarchy needlessly, as widget size has no bearing on the tree hierarchy.

The tree hierarchy's size only changes when items are added/removed, open/closed, label contents or font sizes changed, margins changed, etc.
This calculation involves walking the entire tree from top to bottom, potentially a slow calculation if the tree has many items (potentially hundreds of thousands), and should therefore be called sparingly. For this reason, `recalc_tree()` is used as a way to schedule calculation when changes affect the tree hierarchy's size.

Apps may want to call this method directly if the app makes changes to the tree's geometry, then immediately needs to work with the tree's new dimensions before an actual redraw (and recal) occurs. (This use by an app should only rarely be needed)

12.152.2.5 `callback_item()` [1/2]

```c
Fl_Tree_Item * Fl_Tree::callback_item ( )
```

Gets the item that caused the callback.

The `callback()` can use this value to see which item changed.

12.152.2.6 `callback_item()` [2/2]

```c
void Fl_Tree::callback_item ( Fl_Tree_Item * item )
```

Sets the item that was changed for this callback.

Used internally to pass the item that invoked the callback.

12.152.2.7 `callback_reason()` [1/2]

```c
Fl_Tree_Reason Fl_Tree::callback_reason ( ) const
```

Gets the reason for this callback.

The `callback()` can use this value to see why it was called. Example:

```c
  void MyTreeCallback(Fl_Widget *w, void *userdata) {
    Fl_Tree *tree = (Fl_Tree*)w;
    Fl_Tree_Item *item = tree->callback_item(); // the item changed (can be NULL if more than one item was changed!)
    switch ( tree->callback_reason() ) { // reason callback was invoked
      case FL_TREE_REASON_OPENED: ..item was opened..
      case FL_TREE_REASON_CLOSED: ..item was closed..
      case FL_TREE_REASON_SELECTED: ..item was selected..
      case FL_TREE_REASON_RESELECTED: ..item was reselected (double-clicked, etc)..
      case FL_TREE_REASON_DESELECTED: ..item was deselected..
    }
  }
```

See also

`item_reselect_mode()` – enables `FL_TREE_REASON_RESELECTED` events

12.152.2.8 `callback_reason()` [2/2]

```c
void Fl_Tree::callback_reason ( Fl_Tree_Reason reason )
```

Sets the reason for this callback.

Used internally to pass the reason the callback was invoked.

12.152.2.9 `clear()`

```c
void Fl_Tree::clear ( void )
```

Clear the entire tree's children, including the root.

The tree will be left completely empty.

---

Generated by Doxygen
12.152.2.10  clear_children()

    void Fl_Tree::clear_children (  
        Fl_Tree_Item * item  )

Clear all the children for 'item'.
Item may not be NULL.

12.152.2.11  close() [1/2]

    int Fl_Tree::close (  
        const char * path,  
        int docallback = 1 )

Closes the item specified by 'path'.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. close("Holidays/12\25\/2010").
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the
reason the callback was called.

Parameters

| in | path              | the tree item's pathname (e.g. "Flintstones/Fred") |
|    | docallback        | A flag that determines if the callback() is invoked or not: |
|    |                   | • 0 - callback() is not invoked |
|    |                   | • 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_CLOSED |

Returns

• 1 – OK: item closed
• 0 – OK: item was already closed, no change
• -1 – ERROR: item was not found

See also

   open(), close(), is_open(), is_close(), callback_item(), callback_reason()

12.152.2.12  close() [2/2]

    int Fl_Tree::close (  
        Fl_Tree_Item * item,  
        int docallback = 1 )

Closes the specified 'item'.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the
reason the callback was called.

Parameters

| in | item              | the item to be closed. Must not be NULL. |
|    | docallback        | A flag that determines if the callback() is invoked or not: |
|    |                   | • 0 - callback() is not invoked |
|    |                   | • 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_CLOSED |
Returns

- 1 – item was closed
- 0 – item was already closed, no change

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

12.152.2.13 closeicon() [1/2]

Fl_Image * Fl_Tree::closeicon ( ) const
Returns the icon to be used as the 'close' icon.
If none was set, the internal default is returned, a simple '[ ]' icon.

12.152.2.14 closeicon() [2/2]

void Fl_Tree::closeicon ( Fl_Image * val )
Sets the icon to be used as the 'close' icon.
This overrides the built in default '[ ]' icon.

Parameters

| in | val | – The new image, or zero to use the default '[ ]' icon. |

12.152.2.15 connectorstyle()

void Fl_Tree::connectorstyle ( Fl_Tree_Connector val )
Sets the line drawing style for inter-connecting items.
See Fl_Tree_Connector for possible values.

12.152.2.16 deselect() [1/2]

int Fl_Tree::deselect ( const char * path,
   int docallback = 1 )
Deselect an item specified by 'path'.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. deselect("← Holidays/12\25\2010").
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

| in | path | – the tree item’s pathname (e.g. "Flintstones/Fred") |
| in | docallback | – A flag that determines if the callback() is invoked or not: |
|     |             | • 0 - the callback() is not invoked |
|     |             | • 1 - the callback() is invoked if item changed state (default), callback_reason() will be FL_TREE_REASON_DESELECTED |
Returns

- 1 - OK: item's state was changed
- 0 - OK: item was already deselected, no change was made
- -1 - ERROR: item was not found

12.152.2.17 deselect() [2/2]

```c
int Fl_Tree::deselect (Fl_Tree_Item *item, int docallback = 1)
```

Deselect the specified `item`.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling `redraw()` if anything changed.
The callback can use `callback_item()` and `callback_reason()` respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>the item to be deselected. Must not be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>the callback() is invoked if item changed state (default), callback_reason() will be FL_TREE_REASON_DESELECTED</td>
</tr>
</tbody>
</table>

Returns

- 0 - item was already deselected, no change was made
- 1 - item's state was changed

12.152.2.18 deselect_all()

```c
int Fl_Tree::deselect_all (Fl_Tree_Item *item = 0, int docallback = 1)
```

Deselect 'item' and all its children.
If item is NULL, first() is used.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling `redraw()` if anything changed.
The callback can use `callback_item()` and `callback_reason()` respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item that will be deselected (along with all its children). If NULL, first() is used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>the callback() is invoked for each item that changed state (default), callback_reason() will be FL_TREE_REASON_DESELECTED</td>
</tr>
</tbody>
</table>
12.152 Fl_Tree Class Reference

Returns
Count of how many items were actually changed to the deselected state.

12.152.2.19 display()

void Fl_Tree::display (Fl_Tree_Item * item)
Displays 'item', scrolling the tree as necessary.

Parameters

| in      | item | The item to be displayed. If NULL, first() is used. |

12.152.2.20 displayed()

int Fl_Tree::displayed (Fl_Tree_Item * item)
See if 'item' is currently displayed on-screen (visible within the widget).
This can be used to detect if the item is scrolled off-screen. Checks to see if the item's vertical position is within the
top and bottom edges of the display window. This does NOT take into account the hide() / show() or open() / close() status of the item.

Parameters

| in      | item | The item to be checked. If NULL, first() is used. |

Returns
1 if displayed, 0 if scrolled off screen or no items are in tree.

12.152.2.21 draw()

void Fl_Tree::draw ()
[protected], [virtual]
Standard FLTK draw() method, handles drawing the tree widget.
Reimplemented from Fl_Group.

12.152.2.22 extend_selection()

int Fl_Tree::extend_selection (Fl_Tree_Item * from,
Fl_Tree_Item * to,
int val = 1,
bool visible = false)
Extend a selection between 'from' and 'to' depending on 'visible'.
Similar to the more efficient extend_selection_dir(Fl_Tree_Item*,Fl_Tree_Item*,int dir,int val,bool vis) method, but direction (up or down) doesn't need to be known.
We're less efficient because we search the tree for to/from, then operate on items in between. The more efficient method avoids the "search", but necessitates a direction to be specified to find 'to'.
Used by SHIFT-click to extend a selection between two items inclusive.
Handles calling redraw() if anything changed.

Parameters

| in      | from | Starting item |
| in      | to   | Ending item   |
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
<th>Select or deselect items (0=deselect, 1=select, 2=toggle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>visible</td>
<td>true=affect only open(), visible items, false=affect open or closed items (default)</td>
</tr>
</tbody>
</table>

### Returns

The number of items whose selection states were changed, if any.

### Version

1.3.3 ABI feature

#### 12.152.23 `extend_selection_dir()`

```cpp
def int Fl_Tree::extend_selection_dir (Fl_Tree_Item *from, Fl_Tree_Item *to, int dir, int val, bool visible)
```

Extend the selection between and including 'from' and 'to' depending on direction 'dir', 'val', and 'visible'.

**Efficient:** does not walk entire tree; starts with 'from' and stops at 'to' while moving in direction 'dir'. Dir must be specified though.

- If dir cannot be known in advance, such as during SHIFT-click operations, the method `extend_selection(Fl_Tree_Item*, Fl_Tree_Item*, int, bool)` should be used.
- Handles calling `redraw()` if anything changed.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>from</th>
<th>Starting item</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>to</td>
<td>Ending item</td>
</tr>
<tr>
<td>in</td>
<td>dir</td>
<td>Direction to extend selection (FL_Up or FL_Down)</td>
</tr>
<tr>
<td>in</td>
<td>val</td>
<td>0=deselect, 1=select, 2=toggle</td>
</tr>
<tr>
<td>in</td>
<td>visible</td>
<td>true=affect only open(), visible items, false=affect open or closed items (default)</td>
</tr>
</tbody>
</table>

### Returns

The number of items whose selection states were changed, if any.

### Version

1.3.3

#### 12.152.24 `find_clicked()`

```cpp
def const Fl_Tree_Item * Fl_Tree::find_clicked (int yonly = 0) const
```

Find the item that was last clicked on.

- You should use `callback_item()` instead, which is fast, and is meant to be used within a callback to determine the item clicked.
- This method walks the entire tree looking for the first item that is under the mouse. (The value of the 'yonly' flag affects whether both x and y events are checked, or just y)
Use this method /only/ if you've subclassed Fl_Tree, and are receiving events before Fl_Tree has been able to process and update callback_item().

Parameters

| in | yonly | – 0: check both event's X and Y values. – 1: only check event's Y value, don't care about X. |

Returns

The item clicked, or NULL if no item was under the current event.

Version

1.3.0
1.3.3 ABI feature: added yonly parameter

12.152.2.25 find_item()

const Fl_Tree_Item * Fl_Tree::find_item ( const char * path ) const

Find the item, given a menu style path, e.g. "/Parent/Child/item". There is both a const and non-const version of this method. Const version allows pure const methods to use this method to do lookups without causing compiler errors.

To specify items or submenus that contain slashes ("/" or ") use an escape character to protect them, e.g.

```c
    tree->add("/Holidays/Photos/12\25\2010"); // Adds item "12/25/2010"
    tree->add("/Pathnames/c:\\Program Files\\MyApp"); // Adds item "c:\Program Files\MyApp"
```

Parameters

| in | path | – the tree item's pathname to be found (e.g. "Flintstones/Fred") |

Returns

The item, or NULL if not found.

See also

item_pathname()

12.152.2.26 first()

Fl_Tree_Item * Fl_Tree::first ( )

Returns the first item in the tree, or 0 if none.

Use this to walk the tree in the forward direction, e.g.

```c
    for ( Fl_Tree_Item *item = tree->first(); item; item = tree->next(item) )
    printf("Item: %s\n", item->label());
```

Returns

First item in tree, or 0 if none (tree empty).

See also

first(), next(), last(), prev()
12.152.2.27 first_selected_item()

Fl_Tree_Item * Fl_Tree::first_selected_item ( )
Returns the first selected item in the tree.
Use this to walk the tree from top to bottom looking for all the selected items, e.g.

```c
// Walk tree forward, from top to bottom
for ( Fl_Tree_Item *i=tree->first_selected_item(); i; i=tree->next_selected_item(i) )
    printf("Selected item: %s\n", i->label());
```

Returns
The first selected item, or 0 if none.

See also
first_selected_item(), last_selected_item(), next_selected_item()

12.152.2.28 first_visible()

Fl_Tree_Item * Fl_Tree::first_visible ( )
Returns the first open(), visible item in the tree, or 0 if none.

Deprecated in 1.3.3 ABI – use first_visible_item() instead.

12.152.2.29 first_visible_item()

Fl_Tree_Item * Fl_Tree::first_visible_item ( )
Returns the first open(), visible item in the tree, or 0 if none.

Returns
First visible item in tree, or 0 if none.

See also
first_visible_item(), last_visible_item(), next_visible_item()

Version
1.3.3

12.152.2.30 get_selected_items()

int Fl_Tree::get_selected_items ( Fl_Tree_Item_Array & ret_items )
Returns the currently selected items as an array of 'ret_items'.
Example:

```c
// Get selected items as an array
Fl_Tree_Item_Array items;
tree->get_selected_items(items);
// Manipulate the returned array
for ( int t=0; t<items.total(); t++ )
    Fl_Tree_Item *item = items[t];
    // do stuff with each selected item...
```
Parameters

<table>
<thead>
<tr>
<th></th>
<th>out</th>
<th>ret_items</th>
<th>The returned array of selected items.</th>
</tr>
</thead>
</table>

Returns

The number of items in the returned array.

See also

first_selected_item(), next_selected_item()

Version

1.3.3 ABI feature

12.152.2.31 handle()

int Fl_Tree::handle (  
    int e ) [virtual]

Standard FLTK event handler for this widget.

Todo add Fl_Widget_Tracker (see Fl_Browser_cxx::handle())

Reimplemented from Fl_Group.

12.152.2.32 hposition() [1/2]

int Fl_Tree::hposition ( ) const

Returns the horizontal scroll position as a pixel offset.
The position returned is how many pixels of the tree are scrolled off the left edge of the screen.

See also

hposition(int), vposition(), vposition(int)

Note

Must be using FLTK ABI 1.3.3 or higher for this to be effective.

12.152.2.33 hposition() [2/2]

void Fl_Tree::hposition (  
    int pos )

Sets the horizontal scroll offset to position 'pos'.
The position is how many pixels of the tree are scrolled off the left edge of the screen.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>pos</th>
<th>The vertical position (in pixels) to scroll the tree to.</th>
</tr>
</thead>
</table>

See also

hposition(), vposition(), vposition(int)

Note

Must be using FLTK ABI 1.3.3 or higher for this to be effective.
12.152.2.34 insert()

**Fl_Tree_Item** * Fl_Tree::insert (  
  **Fl_Tree_Item** * item,  
  const char * name,  
  int pos )  
Insert a new item 'name' into 'item's children at position 'pos'.
If pos is out of range the new item is
  • prepended if pos < 0 or  
  • appended if pos > item->children().

Note: pos == children() is not considered out of range: the item is appended to the child list.

Example:

```
    tree->add("Aaa/000"); // "000" is index 0 in Aaa's children
    tree->add("Aaa/111"); // "111" is index 1 in Aaa's children
    tree->add("Aaa/222"); // "222" is index 2 in Aaa's children
    // How to use insert() to insert a new item between Aaa/111 + Aaa/222
    Fl_Tree_Item *item = tree->find_item("Aaa"); // get parent item Aaa
    if (item) tree->insert(item, "New item", 2); // insert as a child of Aaa at index #2
```

Parameters

<table>
<thead>
<tr>
<th>in item</th>
<th>The existing item to insert new child into. Must not be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in name</td>
<td>The label for the new item</td>
</tr>
<tr>
<td>in pos</td>
<td>The position of the new item in the child list</td>
</tr>
</tbody>
</table>

Returns

The new item added.

See also

insert_above()

12.152.2.35 insert_above()

**Fl_Tree_Item** * Fl_Tree::insert_above (  
  **Fl_Tree_Item** * above,  
  const char * name )  
Inserts a new item 'name' above the specified Fl_Tree_Item 'above'.

Example:

```
    tree->add("Aaa/000"); // "000" is index 0 in Aaa's children
    tree->add("Aaa/111"); // "111" is index 1 in Aaa's children
    tree->add("Aaa/222"); // "222" is index 2 in Aaa's children
    // How to use insert_above() to insert a new item above Aaa/222
    Fl_Tree_Item *item = tree->find_item("Aaa/222"); // get item Aaa/222
    if (item) tree->insert_above(item, "New item"); // insert new item above it
```

Parameters

<table>
<thead>
<tr>
<th>in above</th>
<th>– the item above which to insert the new item. Must not be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in name</td>
<td>– the name of the new item</td>
</tr>
</tbody>
</table>
Returns

The new item added, or 0 if 'above' could not be found.

See also

insert()

12.152.2.36  is_close() [1/2]

int Fl_Tree::is_close ( const char * path ) const

See if item specified by 'path' is closed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. is_close("Holidays/12\25\2010").

Parameters

| in   | path     | - the tree item's pathname (e.g. "Flintstones/Fred") |

Returns

- 1 - OK: item is closed
- 0 - OK: item is open
- -1 - ERROR: item was not found

12.152.2.37  is_close() [2/2]

int Fl_Tree::is_close ( Fl_Tree_Item * item ) const

See if the specified 'item' is closed.

Parameters

| in | item     | - the item to be tested. Must not be NULL. |

Returns

- 1 : item is closed
- 0 : item is open

12.152.2.38  is_hscroll_visible()

int Fl_Tree::is_hscroll_visible ( ) const

See if the horizontal scrollbar is currently visible.

Returns

1 if scrollbar visible, 0 if not.

Note

Must be using FLTK ABI 1.3.3 or higher for this to be effective.
12.152.2.39  **is_open() [1/2]**

```cpp
int Fl_Tree::is_open (  
    const char * path ) const  
```

**See if item specified by 'path' is open.**
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. `is_open("←Holidays/12\25\2010")`.
Items that are 'open' are themselves not necessarily visible; one of the item's parents might be closed.

**Parameters**

- **in path** – the tree item's pathname (e.g. "Flintstones/Fred")

**Returns**

- 1 - OK: item is open
- 0 - OK: item is closed
- -1 - ERROR: item was not found

**See also**

`Fl_Tree_Item::visible_r()`

12.152.2.40  **is_open() [2/2]**

```cpp
int Fl_Tree::is_open (  
    Fl_Tree_Item * item ) const  
```

**See if 'item' is open.**
Items that are 'open' are themselves not necessarily visible; one of the item's parents might be closed.

**Parameters**

- **in item** – the item to be tested. Must not be NULL.

**Returns**

- 1 : item is open
- 0 : item is closed

12.152.2.41  **is_scrollbar()**

```cpp
int Fl_Tree::is_scrollbar (  
    Fl_Widget * w )  
```

See if widget 'w' is one of the Fl_Tree widget's scrollbars.
Use this to skip over the scrollbars when walking the child() array. Example:

```cpp
for ( int i=0; i<tree->children(); i++ ) { // walk children  
    Fl_Widget *w = tree->child(i);  
    if ( tree->is_scrollbar(w) ) continue; // skip scrollbars  
    ..do work here..  
}  
```

**Parameters**

- **in w** – Widget to test
12.152 Fl_Tree Class Reference

Returns

1 if \( w \) is a scrollbar, 0 if not.

**Todo** should be const

### 12.152.2.42 is_selected() [1/2]

```cpp
int Fl_Tree::is_selected (const char * path)
```

See if item specified by 'path' is selected.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. `is_selected("Holidays/12/25/2010").`

**Parameters**

- **path** – the tree item's pathname (e.g. "Flintstones/Fred")

**Returns**

- 1 : item selected
- 0 : item deselected
- -1 : item was not found

### 12.152.43 is_selected() [2/2]

```cpp
int Fl_Tree::is_selected (Fl_Tree_Item * item) const
```

See if the specified 'item' is selected.

**Parameters**

- **item** – the item to be tested. Must not be NULL.

**Returns**

- 1 : item selected
- 0 : item deselected

### 12.152.44 is_vscroll_visible()

```cpp
int Fl_Tree::is_vscroll_visible () const
```

See if the vertical scrollbar is currently visible.

**Returns**

1 if scrollbar visible, 0 if not.

### 12.152.45 item_clicked() [1/2]

```cpp
Fl_Tree_Item * Fl_Tree::item_clicked ()
```

Return the item that was last clicked.
Valid only from within the `callback()`.

**Returns**

The item clicked, or 0 if none. 0 may also be used to indicate several items were clicked/changed.

**Deprecated** in 1.3.3 ABI – use `callback_item()` instead.

Generated by Doxygen
12.152.2.46 item_clicked() [2/2]

void Fl_Tree::item_clicked ( Fl_Tree_Item * item ) [protected]
Set the item that was last clicked.
Should only be used by subclasses needing to change this value. Normally Fl_Tree manages this value.

Deprecated in 1.3.3 ABI – use callback_item() instead.

12.152.2.47 item_draw_mode() [1/3]

Fl_Tree_Item_Draw_Mode Fl_Tree::item_draw_mode () const
Get the 'item draw mode' used for the tree.

Version
1.3.1 ABI feature

12.152.2.48 item_draw_mode() [2/3]

void Fl_Tree::item_draw_mode ( Fl_Tree_Item_Draw_Mode mode )
Set the 'item draw mode' used for the tree to 'mode'.
This affects how items in the tree are drawn, such as when a widget() is defined. See Fl_Tree_Item_Draw_Mode for possible values.

Version
1.3.1 ABI feature

12.152.2.49 item_draw_mode() [3/3]

void Fl_Tree::item_draw_mode ( int mode )
Set the 'item draw mode' used for the tree to integer 'mode'.
This affects how items in the tree are drawn, such as when a widget() is defined. See Fl_Tree_Item_Draw_Mode for possible values.

Version
1.3.1 ABI feature

12.152.2.50 item_labelbgcolor() [1/2]

void Fl_Tree::item_labelbgcolor ( Fl_Color val )
Set the default label background color used for creating new items.
A special case is made for color 0xffffffff (default) which is treated as 'transparent'. To change the background color on a per-item basis, use Fl_Tree_Item::labelbgcolor(Fl_Color)

12.152.2.51 item_labelbgcolor() [2/2]

Fl_Color Fl_Tree::item_labelbgcolor () const
Get the default label background color used for creating new items.
If the color is 0xffffffff, it is 'transparent'.
12.152.2.52  item_labelfgcolor()
void Fl_Tree::item_labelfgcolor (  
    Fl_Color val  
)  
Set the default label foreground color used for creating new items.  
To change the foreground color on a per-item basis, use Fl_Tree_Item::labelfgcolor(Fl_Color)

12.152.2.53  item_labelfont()
void Fl_Tree::item_labelfont (  
    Fl_Font val  
)  
Set the default font face used for creating new items.  
To change the font face on a per-item basis, use Fl_Tree_Item::labelfont(Fl_Font)

12.152.2.54  item_labelsize()
void Fl_Tree::item_labelsize (  
    Fl_Fontsize val  
)  
Set the default label font size used for creating new items.  
To change the font size on a per-item basis, use Fl_Tree_Item::labelsize(Fl_Fontsize)

12.152.2.55  item_pathname()
int Fl_Tree::item_pathname (  
    char * pathname,  
    int pathnamelen,  
    const Fl_Tree_Item * item ) const  
Return 'pathname' of size 'pathnamelen' for the specified 'item'.  
If 'item' is NULL, root() is used.  
The tree's root will be included in the pathname if showroot() is on.  
Menu items or submenus that contain slashes ('/' or '\') in their names will be escaped with a backslash.  
This is symmetrical with the add() function which uses the same escape pattern to set names.

Parameters

| out | pathname | The string to use to return the pathname |
| in  | pathnamelen | The maximum length of the string (including NULL). Must not be zero. |
| in  | item       | The item whose pathname is to be returned. |

Returns

- 0 : OK (pathname returns the item's pathname)
- -1 : item not found (pathname="")
- -2 : pathname not large enough (pathname="")

See also

find_item()

12.152.2.56  item_reselect_mode()[1/2]
Fl_Tree_Item_Reselect_Mode Fl_Tree::item_reselect_mode ( ) const  
Returns the current item re/selection mode.

Version

1.3.1 ABI feature
item_reselect_mode() [2/2]

```c
void Fl_Tree::item_reselect_mode ( Fl_Tree_Item_Reselect_Mode mode )
```

Sets the item re/selection mode.
See `Fl_Tree_Item_Reselect_Mode` for possible values.

**Version**

1.3.1 ABI feature

last()

```c
Fl_Tree_Item * Fl_Tree::last ( )
```

Returns the last item in the tree.
This can be used to walk the tree in reverse, e.g.

```c
for ( Fl_Tree_Item *item = tree->last(); item; item = tree->prev() )
   printf("Item: %s\n", item->label());
```

Returns

Last item in the tree, or 0 if none (tree empty).

See also

`first(), next(), last(), prev()`

last_selected_item()

```c
Fl_Tree_Item * Fl_Tree::last_selected_item ( )
```

Returns the last selected item in the tree.
Use this to walk the tree in reverse from bottom to top looking for all the selected items, e.g.

```c
for ( Fl_Tree_Item *i=tree->last_selected_item(); i; i=tree->next_selected_item(i, FL_Up) )
   printf("Selected item: %s\n", i->label());
```

Returns

The last selected item, or 0 if none.

See also

`first_selected_item(), last_selected_item(), next_selected_item()`

Version

1.3.3

last_visible()

```c
Fl_Tree_Item * Fl_Tree::last_visible ( )
```

Returns the last open(), visible item in the tree.

**Deprecated** in 1.3.3 – use `last_visible_item()` instead.
12.152.2.61 last_visible_item()

Fl_Tree_Item * Fl_Tree::last_visible_item ( )

Returns the last open(), visible item in the tree.

Returns

Last visible item in the tree, or 0 if none.

See also

first_visible_item(), last_visible_item(), next_visible_item()

Version

1.3.3

12.152.2.62 load()

void Fl_Tree::load ( class Fl_Preferences & prefs )

Load FLTK preferences.

Read a preferences database into the tree widget.

A preferences database is a hierarchical collection of data which can be directly loaded into the tree view for inspection.

Parameters

| in  | prefs | the Fl_Preferences database |

12.152.2.63 next()

Fl_Tree_Item * Fl_Tree::next ( Fl_Tree_Item * item = 0 )

Return the next item after 'item', or 0 if no more items.

Use this code to walk the entire tree:

```c
for ( Fl_Tree_Item *i = tree->first(); i; i = tree->next(i) )
    printf("Item: %s\n", i->label());
```

Parameters

| in  | item | The item to use to find the next item. If NULL, returns 0. |

Returns

Next item in tree, or 0 if at last item.

See also

first(), next(), last(), prev()

12.152.2.64 next_item()

Fl_Tree_Item * Fl_Tree::next_item ( Fl_Tree_Item * item, 0 )

Parameters

| in  | item | |

Returns

Next item in tree, or 0 if at last item.

See also

first(), next(), last(), prev()
int dir = FL_Down,
bool visible = false)

Returns next item after 'item' in direction 'dir' depending on 'visible'.
Next item will be above (if dir==FL_Up) or below (if dir==FL_Down). If 'visible' is true, only items whose parents are open() will be returned. If 'visible' is false, even items whose parents are close()ed will be returned.
If item is 0, the return value will be the result of this truth table:

<table>
<thead>
<tr>
<th>visible=true</th>
<th>visible=false</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir=FL_Up</td>
<td>last_visible_item()</td>
</tr>
<tr>
<td>dir=FL_Down</td>
<td>first_visible_item()</td>
</tr>
</tbody>
</table>

Example use:

for ( Fl_Tree_Item *i=tree->first_visible_item(); i; i=tree->next_item(i, FL_Down, true) )
    printf("Item: %s\n", i->label());
for ( Fl_Tree_Item *i=tree->last_visible_item(); i; i=tree->next_item(i, FL_Up, true) )
    printf("Item: %s\n", i->label());
for ( Fl_Tree_Item *i=tree->first(); i; i=tree->next_item(i, FL_Down, false) )
    printf("Item: %s\n", i->label());
for ( Fl_Tree_Item *i=tree->last(); i; i=tree->next_item(i, FL_Up, false) )
    printf("Item: %s\n", i->label());

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item to use to find the next item. If NULL, returns 0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dir</td>
<td>Can be FL_Up or FL_Down (default=FL_Down or 'next')</td>
</tr>
<tr>
<td>in</td>
<td>visible</td>
<td>true=return only open(), visible items, false=return open or closed items (default)</td>
</tr>
</tbody>
</table>

Returns

Next item in tree in the direction and visibility specified, or 0 if no more items of specified visibility in that direction.

See also

first(), last(), next(),
first_visible_item(), last_visible_item(), next_visible_item(),
first_selected_item(), last_selected_item(), next_selected_item()

Version

1.3.3

12.152.2.65 next_selected_item()

Fl_Tree_Item * Fl_Tree::next_selected_item (  
    Fl_Tree_Item * item = 0,
    int dir = FL_Down )

Returns the next selected item above or below 'item', depending on 'dir'.
If 'item' is 0, search starts at either first() or last(), depending on 'dir': first() if 'dir' is FL_Down (default), last() if 'dir' is FL_Up.
Use this to walk the tree looking for all the selected items, e.g.
// Walk down the tree (forwards)
for ( Fl_Tree_Item *i=tree->first_selected_item(); i; i=tree->next_selected_item(i, FL_Down) )
  printf("Item: %s\n", i->label());

// Walk up the tree (backwards)
for ( Fl_Tree_Item *i=tree->last_selected_item(); i; i=tree->next_selected_item(i, FL_Up) )
  printf("Item: %s\n", i->label());

Parameters

| in  | item | The item above or below which we'll find the next selected item. If NULL, first() is used if FL_Down, last() if FL_Up. (default=NULL) |
| in  | dir  | The direction to go. FL_Up for moving up the tree, FL_Down for down the tree (default) |

Returns

The next selected item, or 0 if there are no more selected items.

See also

first_selected_item(), last_selected_item(), next_selected_item()

Version

1.3.3

12.152.2.66 next_visible_item()

Fl_Tree_Item * Fl_Tree::next_visible_item ( Fl_Tree_Item *item, int dir )

Returns next open(), visible item above (dir==FL_Up) or below (dir==FL_Down) the specified 'item', or 0 if no more items. If 'item' is 0, returns last() if 'dir' is FL_Up, or first() if dir is FL_Down.

Parameters

| in  | item  | The item above/below which we'll find the next visible item |
| in  | dir   | The direction to search. Can be FL_Up or FL_Down. |

Returns

The item found, or 0 if there's no visible items above/below the specified item.

Version

1.3.3
12.152.2.67 open() [1/2]

int Fl_Tree::open (  
    const char * path,  
    int docallback = 1 )

Opens the item specified by 'path'.
This causes the item's children (if any) to be shown.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. open("Holidays/12\25\2010").
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>– the tree item's pathname (e.g. &quot;Flintstones/Fred&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>– A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_OPENED</td>
</tr>
</tbody>
</table>

Returns

• 1 – OK: item opened
• 0 – OK: item was already open, no change
• -1 – ERROR: item was not found

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

12.152.2.68 open() [2/2]

int Fl_Tree::open (  
    Fl_Tree_Item * item,  
    int docallback = 1 )

Open the specified 'item'.
This causes the item's children (if any) to be shown.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>– the item to be opened. Must not be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>– A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_OPENED</td>
</tr>
</tbody>
</table>
Returns

- 1 – item was opened
- 0 – item was already open, no change

See also

`open()`, `close()`, `is_open()`, `is_close()`, `callback_item()`, `callback_reason()`

### 12.152.2.69 open_toggle()

```cpp
void Fl_Tree::open_toggle(
    Fl_Tree_Item *item,
    int docallback = 1
)
```

Toggle the open state of 'item'.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling `redraw()` if anything changed.
The callback can use `callback_item()` and `callback_reason()` respectively to determine the item changed and the reason the callback was called.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>the item whose open state is to be toggled. Must not be NULL.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>in</th>
<th>docallback</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td>0 - callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td>1 - callback() is invoked (default), callback_reason() will be either FL_TREE_REASON_OPENED or FL_TREE_REASON_CLOSED</td>
</tr>
</tbody>
</table>

See also

`open()`, `close()`, `is_open()`, `is_close()`, `callback_item()`, `callback_reason()`

### 12.152.2.70 openicon() [1/2]

```cpp
Fl_Image * Fl_Tree::openicon() const
```

Returns the icon to be used as the 'open' icon.
If none was set, the internal default is returned, a simple '[+] icon.

### 12.152.2.71 openicon() [2/2]

```cpp
void Fl_Tree::openicon(
    Fl_Image * val
)
```

Sets the icon to be used as the 'open' icon.
This overrides the built in default '[+] icon.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>The new image, or zero to use the default [+] icon.</td>
</tr>
</tbody>
</table>

### 12.152.2.72 prev()

```cpp
Fl_Tree_Item * Fl_Tree::prev(
    Fl_Tree_Item * item = 0
)
```

Return the previous item before 'item', or 0 if no more items.

Generated by Doxygen
This can be used to walk the tree in reverse, e.g.

```c
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->prev(item) )
  printf("Item: %s\n", item->label());
```

Parameters

| in  | item | The item to use to find the previous item. If NULL, returns 0. |

Returns

Previous item in tree, or 0 if at first item.

See also

first(), next(), last(), prev()

12.152.2.73 recalc_tree()

```c
void Fl_Tree::recalc_tree ( )
```

Schedule tree to recalc the entire tree size.

Note

Must be using FLTK ABI 1.3.3 or higher for this to be effective.

12.152.2.74 remove()

```c
int Fl_Tree::remove ( Fl_Tree_Item *item )
```

Remove the specified 'item' from the tree.

item may not be NULL. If it has children, all those are removed too. If item being removed has focus, no item will have focus.

Returns

0 if done, -1 if 'item' not found.

12.152.2.75 resize()

```c
void Fl_Tree::resize ( int X, int Y, int W, int H ) [virtual]
```

Resizes the Fl_Group widget and all of its children.

The Fl_Group widget first resizes itself, and then it moves and resizes all its children according to the rules documented for Fl_Group::resizable(Fl_Widget*)

See also

Fl_Group::resizable(Fl_Widget*)

Fl_Group::resizable()

Fl_Widget::resize(int,int,int,int)

Reimplemented from Fl_Group.
12.152.2.76 root()

void Fl_Tree::root (  
    Fl_Tree_Item * newitem  
)  
Sets the root item to 'newitem'.
If a root item already exists, clear() is called first to clear it before replacing it with newitem.
Use this to install a custom item (derived from Fl_Tree_Item) as the root of the tree. This allows the derived class to implement custom drawing by overriding Fl_Tree_Item::draw_item_content().

Version

1.3.3

12.152.2.77 root_label()

void Fl_Tree::root_label (  
    const char * new_label  
)  
Set the label for the root item to 'new_label'.
Makes an internally managed copy of 'new_label'.

12.152.2.78 scrollbar_size() [1/2]

int Fl_Tree::scrollbar_size ( ) const  
Gets the default size of scrollbars' troughs for this widget in pixels.
If this value is zero (default), this widget will use the global Fl::scrollbar_size() value as the scrollbar's width.

Returns

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int)

12.152.2.79 scrollbar_size() [2/2]

void Fl_Tree::scrollbar_size (  
    int size  
)  
Sets the pixel size of the scrollbars' troughs to 'size' for this widget, in pixels.
Normally you should not need this method, and should use the global Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, and is the default behavior.
Normally this is what you want.
Only use this method if you really need to override just THIS instance of the widget's scrollbar size. (This need should be rare.)
Setting size to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.

Parameters

| in  | size | Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size() |

See also

Fl::scrollbar_size()

12.152.2.80 select() [1/2]

int Fl_Tree::select (  
    const char * path,  
)
int docallback = 1 )

Select the item specified by 'path'.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. select("Holidays/12\25\2010").
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>– the tree item's pathname (e.g. &quot;Flintstones/Fred&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>– A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 - the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - the callback() is invoked if item changed state (default), callback_reason() will be FL_TREE_REASON_SELECTED</td>
</tr>
</tbody>
</table>

Returns

- 1 : OK: item's state was changed
- 0 : OK: item was already selected, no change was made
- -1 : ERROR: item was not found

12.152.2.81 select() [2/2]

int Fl_Tree::select ( Fl_Tree_Item * item, int docallback = 1 )

Select the specified 'item'.
Use 'deselect()' to deselect it.
Invokes the callback depending on the value of optional parameter docallback.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>– the item to be selected. Must not be NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>– A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 - the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - the callback() is invoked if item changed state, callback_reason() will be FL_TREE_REASON_SELECTED</td>
</tr>
</tbody>
</table>

Returns

- 1 - item's state was changed
- 0 - item was already selected, no change was made

12.152.2.82 select_all()

int Fl_Tree::select_all ( Fl_Tree_Item * item = 0, int docallback = 1 )
Select 'item' and all its children.
If item is NULL, first() is used.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item that will be selected (along with all its children). If NULL, first() is used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - the callback() is invoked for each item that changed state (default),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>callback_reason() will be FL_TREE_REASON_SELECTED</td>
</tr>
</tbody>
</table>

Returns

Count of how many items were actually changed to the selected state.

12.152.2.83  select_only()

int Fl_Tree::select_only (Fl_Tree_Item ∗ selitem, int docallback = 1)

Select only the specified item, deselecting all others that might be selected.
If 'selitem' is 0, first() is used.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>selitem</th>
<th>The item to be selected. If NULL, first() is used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - the callback() is invoked for each item that changed state (default),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>callback_reason() will be either FL_TREE_REASON_SELECTED or FL_TREE_REASON_DESELECTED</td>
</tr>
</tbody>
</table>

Returns

The number of items whose selection states were changed, if any.

12.152.2.84  select_toggle()

void Fl_Tree::select_toggle (Fl_Tree_Item ∗ item, int docallback = 1)

Toggle the select state of the specified 'item'.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use `callback_item()` and `callback_reason()` respectively to determine the item changed and the reason the callback was called.

### Parameters

| in | item | – the item to be selected. Must not be NULL. |
| in | docallback | – A flag that determines if the `callback()` is invoked or not: |
| | | • 0 - the `callback()` is not invoked |
| | | • 1 - the `callback()` is invoked (default), `callback_reason()` will be either `FL_TREE_REASON_SELECTED` or `FL_TREE_REASON_DESELECTED` |

#### 12.152.2.85 selectbox() [1/2]

```cpp
Fl_Boxtype Fl_Tree::selectbox ( ) const
```

Sets the style of box used to draw selected items.
This is an fltk `Fl_Boxtype`. The default is influenced by FLTK's current `Fl::scheme()`.

#### 12.152.2.86 selectbox() [2/2]

```cpp
void Fl_Tree::selectbox ( Fl_Boxtype val )
```

Gets the style of box used to draw selected items.
This is an fltk `Fl_Boxtype`. The default is influenced by FLTK's current `Fl::scheme()`.

#### 12.152.2.87 selectmode() [1/2]

```cpp
Fl_Tree_Select Fl_Tree::selectmode ( ) const
```

Gets the tree's current selection mode.
See `Fl_Tree_Select` for possible values.

#### 12.152.2.88 selectmode() [2/2]

```cpp
void Fl_Tree::selectmode ( Fl_Tree_Select val )
```

Sets the tree's selection mode.
See `Fl_Tree_Select` for possible values.

#### 12.152.2.89 set_item_focus()

```cpp
void Fl_Tree::set_item_focus ( Fl_Tree_Item * item )
```

Set the item that currently should have keyboard focus.
Handles calling `redraw()` to update the focus box (if it is visible).

### Parameters

| in | item | The item that should take focus. If NULL, none will have focus. |

#### 12.152.2.90 show_item() [1/2]

```cpp
void Fl_Tree::show_item ( Fl_Tree_Item * item )
```

Adjust the vertical scrollbar to show `item` at the top of the display IF it is currently off-screen (for instance `show_item_top()`).
If it is already on-screen, no change is made.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>item</code></td>
<td>The item to be shown. If NULL, <code>first()</code> is used.</td>
</tr>
</tbody>
</table>

See also

- `show_item_top()`, `show_item_middle()`, `show_item_bottom()`

### 12.152.2.91 show_item()

```cpp
void Fl_Tree::show_item (Fl_Tree_Item *item, int yoff)
```

Adjust the vertical scrollbar so that 'item' is visible 'yoff' pixels from the top of the Fl_Tree widget's display. For instance, `yoff=0` will position the item at the top. If `yoff` is larger than the vertical scrollbar's limit, the value will be clipped. So if `yoff=100`, but scrollbar's max is 50, then 50 will be used.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>item</code></td>
<td>The item to be shown. If NULL, <code>first()</code> is used.</td>
</tr>
<tr>
<td><code>yoff</code></td>
<td>The pixel offset from the top for the displayed position.</td>
</tr>
</tbody>
</table>

See also

- `show_item_top()`, `show_item_middle()`, `show_item_bottom()`

### 12.152.2.92 show_item_bottom()

```cpp
void Fl_Tree::show_item_bottom (Fl_Tree_Item *item)
```

Adjust the vertical scrollbar so that 'item' is at the bottom of the display.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>item</code></td>
<td>The item to be shown. If NULL, <code>first()</code> is used.</td>
</tr>
</tbody>
</table>

### 12.152.2.93 show_item_middle()

```cpp
void Fl_Tree::show_item_middle (Fl_Tree_Item *item)
```

Adjust the vertical scrollbar so that 'item' is in the middle of the display.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>item</code></td>
<td>The item to be shown. If NULL, <code>first()</code> is used.</td>
</tr>
</tbody>
</table>

### 12.152.2.94 show_item_top()

```cpp
void Fl_Tree::show_item_top (Fl_Tree_Item *item)
```

Adjust the vertical scrollbar so that 'item' is at the top of the display.
Parameters

| in | item | The item to be shown. If NULL, first() is used. |

**12.152.2.95 show_self()**

```cpp
void Fl_Tree::show_self ( )
```

Print the tree as 'ascii art' to stdout.
Used mainly for debugging.

Todo should be const

**Version**

1.3.0

**12.152.2.96 showcollapse() [1/2]**

```cpp
int Fl_Tree::showcollapse ( ) const
```

Returns 1 if the collapse icon is enabled, 0 if not.

See also

- `showcollapse(int)`

**12.152.2.97 showcollapse() [2/2]**

```cpp
void Fl_Tree::showcollapse ( int val )
```

Set if we should show the collapse icon or not.
If collapse icons are disabled, the user will not be able to interactively collapse items in the tree, unless the application provides some other means via `open()` and `close()`.

Parameters

| in | val | 1: shows collapse icons (default), 0: hides collapse icons. |

**12.152.2.98 showroot()**

```cpp
void Fl_Tree::showroot ( int val )
```

Set if the root item should be shown or not.

Parameters

| in | val | 1 – show the root item (default), 0 – hide the root item. |

**12.152.2.99 sortorder()**

```cpp
Fl_Tree_Sort Fl_Tree::sortorder ( ) const
```

Set the default sort order used when items are added to the tree.
See `Fl_Tree_Sort` for possible values.
12.152.2.100 usericon() [1/2]

```
Fl_Image * Fl_Tree::usericon ( ) const
```

Returns the Fl_Image being used as the default user icon for all newly created items. Returns zero if no icon has been set, which is the default.

12.152.2.101 usericon() [2/2]

```
void Fl_Tree::usericon ( Fl_Image * val )
```

Sets the Fl_Image to be used as the default user icon for all newly created items. If you want to specify user icons on a per-item basis, use Fl_Tree_Item::usericon() instead.

Parameters

- `val`: The new image to be used, or zero to disable user icons.

12.152.2.102 vposition() [1/2]

```
int Fl_Tree::vposition ( ) const
```

Returns the vertical scroll position as a pixel offset. The position returned is how many pixels of the tree are scrolled off the top edge of the screen.

See also

- vposition(int), hposition(), hposition(int)

12.152.2.103 vposition() [2/2]

```
void Fl_Tree::vposition ( int pos )
```

Sets the vertical scroll offset to position 'pos'. The position is how many pixels of the tree are scrolled off the top edge of the screen.

Parameters

- `pos`: The vertical position (in pixels) to scroll the tree to.

See also

- vposition(), hposition(), hposition(int)

The documentation for this class was generated from the following files:

- Fl_Tree.H
- Fl_Tree.cxx

12.153 Fl_Tree_Item Class Reference

Tree widget item.

```
#include <Fl_Tree_Item.H>
```

Public Member Functions

- `void activate (int val=1)`
  
  Change the item's activation state to the optionally specified 'val'.
- `Fl_Tree_Item * add (const Fl_Tree_Prefs &prefs, char **arr)`
Descend into the path specified by 'arr', and add a new child there.

- **FL_Tree_Item** → add (const Fl_Tree_Prefs &prefs, char **arr, FL_Tree_Item *newitem)
  Descend into path specified by 'arr' and add 'newitem' there.

- **FL_Tree_Item** → add (const Fl_Tree_Prefs &prefs, const char *new_label)
  Add a new child to this item with the name 'new_label' and defaults from 'prefs'.

- **FL_Tree_Item** → add (const Fl_Tree_Prefs &prefs, const char *new_label, FL_Tree_Item *newitem)
  Add 'item' as immediate child with 'new_label' and defaults from 'prefs'.

- **FL_Tree_Item** → child (int index)
  Return the child item for the given 'index'.

- const **FL_Tree_Item** → child (int t) const
  Return the const child item for the given 'index'.

- int children () const
  Return the number of children this item has.

- void clear_children ()
  Clear all the children for this item.

- void close ()
  Close this item and all its children.

- void deactivate ()
  Deactivate the item; the callback() won't be invoked when clicked.

- **FL_Tree_Item** → deparent (int index)
  Deparent child at index position 'pos'.

- int depth () const
  Returns how many levels deep this item is in the hierarchy.

- void deselect ()
  Disable the item's selection state.

- int deselect_all ()
  Deselect item and all its children.

- void draw (int X, int &Y, int W, FL_Tree_Item *itemfocus, int &tree_item_xmax, int lastchild=1, int render=1)
  Draw this item and its children.

- virtual int draw_item_content (int render)
  Draw the item content.

- int event_on_collapse_icon (const Fl_Tree_Prefs &prefs) const
  Was the event on the 'collapse' button of this item?

- int event_on_item (const Fl_Tree_Prefs &prefs) const
  Was event anywhere on the item?

- int event_on_label (const Fl_Tree_Prefs &prefs) const
  Was event on the label() of this item?

- int event_on_user_icon (const Fl_Tree_Prefs &prefs) const
  Was the event on the 'user icon' of this item, if any?

- int find_child (const char *name)
  Return the index of the immediate child of this item that has the label 'name'.

- int find_child (FL_Tree_Item *item)
  Find index number for the specified 'item' in the current item's list of children.

- **FL_Tree_Item** → find_child_item (char **arr)
  Non-const version of FL_Tree_Item::find_child_item(char **arr) const.

- const **FL_Tree_Item** → find_child_item (char **arr) const
  Find child item by descending array 'arr' of names.

- **FL_Tree_Item** → find_child_item (const char *name)
  Non-const version of FL_Tree_Item::find_child_item(const char *name) const.

- const **FL_Tree_Item** → find_child_item (const char *name) const
  Return the /immediate/ child of current item that has the label 'name'.
• **Fl_Tree_Item** *find_clicked* (const Fl_Tree_Prefs &prefs, int yonly=0)
  Non-const version of Fl_Tree_Item::find_clicked(const Fl_Tree_Prefs&, int) const.
  `find_clicked` finds the item that the last event was over.

• **Fl_Tree_Item** *find_item* (char **arr)
  Non-const version of Fl_Tree_Item::find_item(char **names) const.
  `find_item` finds an item by descending array of 'names'.

• **Fl_Tree_Item** (const Fl_Tree_Item *o)
  Copy constructor.

• **Fl_Tree_Item** (const Fl_Tree_Prefs &prefs)
  Constructor.

• **Fl_Tree_Item** (Fl_Tree *tree)
  Constructor.

• int **h** () const
  The item's height.

• int has_children () const
  See if this item has children.

• Fl_Tree_Item *insert* (const Fl_Tree_Prefs &prefs, const char *new_label, int pos=0)
  Insert a new item named 'new_label' into current item's children at a specified position 'pos'.

• Fl_Tree_Item *insert_above* (const Fl_Tree_Prefs &prefs, const char *new_label)
  Insert a new item named 'new_label' above this item.

• char is_activated () const
  See if the item is activated.

• char is_active () const
  See if the item is activated. Alias for is_activated().

• int is_close () const
  See if the item is 'closed'.

• int is_open () const
  See if the item is 'open'.

• int is_root () const
  Is this item the root of the tree?

• char is_selected () const
  See if the item is selected.

• int is_visible () const
  See if the item is visible.

• const char * label () const
  Return the label.

• void label (const char *val)
  Set the label to 'name'.

• int label_h () const
  The item's label height.

• int label_w () const
  The item's maximum label width to right edge of Fl_Tree's inner width within scrollbars.

• int label_x () const
  The item's label x position relative to the window.

• int label_y () const
  The item's label y position relative to the window.

• Fl_Color labelbgcolor () const
  Return item's label background text color.

• void labelbgcolor (Fl_Color val)
Set item's label background color.

- **FL_Color** `labelcolor()` const
  
  Return item's label text color. Alias for `labelfgcolor()` const.

  • void `labelcolor(FL_Color val)`
  
  Set item's label text color. Alias for `labelfgcolor(FL_Color)`.

- **FL_Color** `labelfgcolor()` const
  
  Return item's label foreground text color.

  • void `labelfgcolor(FL_Color val)`
  
  Set item's label foreground text color.

- **FL_Font** `labelfont()` const
  
  Get item's label font face.

  • void `labelfont(FL_Font val)`
  
  Set item's label font face.

- **FL_Fontsize** `labelsizesize()` const
  
  Get item's label font size.

  • void `labelsizesize(FL_Fontsize val)`
  
  Set item's label font size.

- int `move(FL_Tree_Item *item, int op=0, int pos=0)`
  
  Move the current item above/below/into the specified `item`, where `op` determines the type of move:

  - int move (int to, int from)
  
  Move an item within its parent using index numbers.

  - int `move_above(FL_Tree_Item *item)`
  
  Move the current item above the specified `item`.

  - int `move_below(FL_Tree_Item *item)`
  
  Move the current item below the specified `item`.

  - int `move_into(FL_Tree_Item *item, int pos=0)`
  
  Parent the current item as a child of the specified `item`.

  - **FL_Tree_Item** * `next()`
  
  Return the next item in the tree.

  - **FL_Tree_Item** * `next_displayed(FL_Tree_Prefs &prefs)`
  
  Same as `next_visible()`.

  - **FL_Tree_Item** * `next_sibling()`
  
  Return this item's next sibling.

  - **FL_Tree_Item** * `next_visible(FL_Tree_Prefs &prefs)`
  
  Return the next `open()`, `visible()` item.

  • void `open()`
  
  Open this item and all its children.

  • void `open_toggle()`
  
  Toggle the item's open/closed state.

  - **FL_Tree_Item** * `parent()`
  
  Return the parent for this item. Returns NULL if we are the root.

  - const **FL_Tree_Item** * `parent()` const
  
  Return the const parent for this item. Returns NULL if we are the root.

  • void `parent(FL_Tree_Item *val)`
  
  Set the parent for this item.

  - const **FL_Tree_Prefs & prefs()** const
  
  Return the parent tree's prefs.

  - **FL_Tree_Item** * `prev()`
  
  Return the previous item in the tree.

  - **FL_Tree_Item** * `prev_displayed(FL_Tree_Prefs &prefs)`
  
  Same as `prev_visible()`.
• `Fl_Tree_Item * prev_sibling ()`
  Return this item's previous sibling.

• `Fl_Tree_Item * prev_visible (Fl_Tree_Prefs &prefs)`
  Return the previous open(), visible() item.

• `int remove_child (const char *new_label)`
  Remove immediate child (and its children) by its label 'name'.

• `int remove_child (Fl_Tree_Item *item)`
  Remove 'item' from the current item's children.

• `int reparent (Fl_Tree_Item *newchild, int index)`
  Reparent specified item as a child of ourself at position 'pos'.

• `Fl_Tree_Item * replace (Fl_Tree_Item *new_item)`
  Replace the current item with a new item.

• `Fl_Tree_Item * replace_child (Fl_Tree_Item *olditem, Fl_Tree_Item *newitem)`
  Replace existing child 'olditem' with 'newitem'.

• `void select (int val=1)`
  Change the item's selection state to the optionally specified 'val'.

• `int select_all ()`
  Select item and all its children.

• `void select_toggle ()`  
  Toggle the item's selection state.

• `void show_self (const char *indent="") const`  
  Print the tree as 'ascii art' to stdout.

• `int swap_children (Fl_Tree_Item *a, Fl_Tree_Item *b)`
  Swap two of our immediate children, given item pointers.

• `void swap_children (int ax, int bx)`
  Swap two of our children, given two child index values 'ax' and 'bx'.

• `Fl_Tree * tree ()`
  Return the tree for this item.

• `const Fl_Tree * tree () const`  
  Return the tree for this item.

• `void update_prev_next (int index)`
  Update our _prev_sibling and _next_sibling pointers to point to neighbors given index as being our current position in the parent's item array.

• `void * user_data () const`  
  Retrieve the user-data value that has been assigned to the item.

• `void user_data (void *data)`
  Set a user-data value for the item.

• `Fl_Image * userdeicon () const`  
  Return the deactivated version of the user icon, if any.

• `void userdeicon (Fl_Image *val)`
  Set the usericon to draw when the item is deactivated.

• `Fl_Image * usericon () const`  
  Get the item's user icon as an Fl_Image. Returns '0' if disabled.

• `void usericon (Fl_Image *val)`
  Set the item's user icon to an Fl_Image.

• `int visible () const`  
  See if the item is visible. Alias for is_visible().

• `int visible_r () const`  
  See if item and all its parents are open() and visible().

• `int w () const`  
  The entire item's width to right edge of Fl_Tree's inner width within scrollbars.
**FL_Widget**

- **widget()** const
  
  Return FLTK widget assigned to this item.

- **void widget(Fl_Widget *val)**
  
  Assign an FLTK widget to this item.

- **int x() const**
  
  The item's x position relative to the window.

- **int y() const**
  
  The item's y position relative to the window.

### Protected Member Functions

- **void _Init(const Fl_Tree_Prefs &prefs, Fl_Tree *tree)**

- **int calc_item_height(const Fl_Tree_Prefs &prefs) const**

  Return the item's 'visible' height.

- **void draw_horizontal_connector(int x1, int x2, int y, const Fl_Tree_Prefs &prefs)**

  Internal: Horizontal connector line based on preference settings.

- **void draw_vertical_connector(int x, int y1, int y2, const Fl_Tree_Prefs &prefs)**

  Internal: Vertical connector line based on preference settings.

- **Fl_Color drawbgcolor() const**

  Returns the recommended background color used for drawing this item.

- **Fl_Color drawfgcolor() const**

  Returns the recommended foreground color used for drawing this item.

- **void hide_widgets()**

  Internal: Hide the FLTK widget() for this item and all children.

- **int is_flag(unsigned short val) const**

  See if flag set. Returns 0 or 1.

- **void recalc_tree()**

  Call this when our geometry is changed.

- **void set_flag(unsigned short flag, int val)**

  Set a flag to an on or off value. val is 0 or 1.

- **void show_widgets()**

  Internal: Show the FLTK widget() for this item and all children.

### 12.153.1 Detailed Description

Tree widget item.

This class is a single tree item, and manages all of the item's attributes. **Fl_Tree_Item** is used by **Fl_Tree**, which is comprised of many instances of **Fl_Tree_Item**.

**Fl_Tree_Item** is hierarchical; it dynamically manages an **Fl_Tree_Item_Array** of children that are themselves instances of **Fl_Tree_Item**. Each item can have zero or more children. When an item has children, **close()** and **open()** can be used to hide or show them.

Items have their own attributes; font size, face, color. Items maintain their own hierarchy of children.

When you make changes to items, you'll need to tell the tree to **redraw()** for the changes to show up.

**New 1.3.3 ABI feature:** You can define custom items by either adding a custom widget to the item with **Fl_Tree_Item::widget()**, or override the **draw_item_content()** method if you want to just redefine how the label is drawn.

The following shows the **Fl_Tree_Item**'s dimensions, useful when overriding the **draw_item_content()** method:
12.153.2 Constructor & Destructor Documentation

12.153.2.1 Fl_Tree_Item() [1/2]

Fl_Tree_Item::Fl_Tree_Item ( const Fl_Tree_Prefs & prefs )

Constructor.
Makes a new instance of Fl_Tree_Item using defaults from 'prefs'.

Deprecated in 1.3.3 ABI – you must use Fl_Tree_Item(Fl_Tree*) for proper horizontal scrollbar behavior.

12.153.2.2 Fl_Tree_Item() [2/2]

Fl_Tree_Item::Fl_Tree_Item ( Fl_Tree * tree )

Constructor.
Makes a new instance of Fl_Tree_Item for 'tree'.
This must be used instead of the older, deprecated Fl_Tree_Item(Fl_Tree_Prefs) constructor for proper horizontal scrollbar calculation.

Version
1.3.3 ABI feature

12.153.3 Member Function Documentation

12.153.3.1 activate()

void Fl_Tree_Item::activate ( int val = 1 ) [inline]

Change the item's activation state to the optionally specified 'val'.
When deactivated, the item will be 'grayed out'; the callback() won’t be invoked if the user clicks on the label. If a widget() is associated with the item, its activation state will be changed as well.
If 'val' is not specified, the item will be activated.

12.153.3.2 add() [1/4]

Fl_Tree_Item * Fl_Tree_Item::add ( const Fl_Tree_Prefs & prefs, char ** arr )

Descend into the path specified by 'arr', and add a new child there.
Should be used only by Fl_Tree's internals. Adds the item based on the value of prefs.sortorder().

Returns
the item added.

Version
1.3.0 release
12.153.3.3  add() [2/4]

```
Fl_Tree_Item * Fl_Tree_Item::add (  
    const Fl_Tree_Prefs & prefs,  
    char ** arr,  
    Fl_Tree_Item * newitem )
```

Descend into path specified by 'arr' and add 'newitem' there.
Should be used only by Fl_Tree's internals. If item is NULL, a new item is created. Adds the item based on the
value of prefs.sortorder().

Returns

the item added.

Version

1.3.3 ABI feature

12.153.3.4  add() [3/4]

```
Fl_Tree_Item * Fl_Tree_Item::add (  
    const Fl_Tree_Prefs & prefs,  
    const char * new_label )
```

Add a new child to this item with the name 'new_label' and defaults from 'prefs'.
An internally managed copy is made of the label string. Adds the item based on the value of prefs.sortorder().

Returns

the item added

Version

1.3.0 release

12.153.3.5  add() [4/4]

```
Fl_Tree_Item * Fl_Tree_Item::add (  
    const Fl_Tree_Prefs & prefs,  
    const char * new_label,  
    Fl_Tree_Item * item )
```

Add 'item' as immediate child with 'new_label' and defaults from 'prefs'.
If 'item' is NULL, a new item is created. An internally managed copy is made of the label string. Adds the item
based on the value of prefs.sortorder().

Returns

the item added

Version

1.3.3

12.153.3.6  calc_item_height()

```
int Fl_Tree_Item::calc_item_height (  
    const Fl_Tree_Prefs & prefs ) const [protected]
```

Return the item's 'visible' height.
Takes into account the item's:

- visibility (if lis_visible(), returns 0)
12.153 Fl_Tree_Item Class Reference

- labelfont() height: if label() != NULL
- widget() height: if widget() != NULL
- openicon() height (if not NULL)
- usericon() height (if not NULL) Does NOT include Fl_Tree::linespacing();

Returns
  maximum pixel height

12.153.3.7 child()

const Fl_Tree_Item * Fl_Tree_Item::child (  
  int t ) const

Return the const child item for the given 'index'.
Return const child item for the specified 'index'.

12.153.3.8 deactivate()

void Fl_Tree_Item::deactivate ( ) [inline]

Deactivate the item; the callback() won't be invoked when clicked.
Same as activate(0)

12.153.3.9 deparent()

Fl_Tree_Item * Fl_Tree_Item::deparent (  
  int pos )

Deparent child at index position 'pos'.
This creates an "orphaned" item that is still allocated, but has no parent or siblings. Normally the caller would want
  to immediately reparent the orphan elsewhere.
A successfully orphaned item will have its parent() and prev_sibling()/next_sibling() set to NULL.

Returns
  • pointer to orphaned item on success
  • NULL on error (could not deparent the item)

See also
  reparent()

12.153.3.10 depth()

int Fl_Tree_Item::depth ( ) const

Returns how many levels deep this item is in the hierarchy.
For instance; root has a depth of zero, and its immediate children would have a depth of 1, and so on. Use e.g. for
determining the horizontal indent of this item during drawing.

12.153.3.11 deselect_all()

int Fl_Tree_Item::deselect_all ( ) [inline]

Deselect item and all its children.
Returns count of how many items were in the 'selected' state, ie. how many items were "changed".
12.153.3.12 draw()

```c
void Fl_Tree_Item::draw (  
    int X,  
    int & Y,  
    int W,  
    Fl_Tree_Item * itemfocus,  
    int & tree_item_xmax,  
    int lastchild = 1,  
    int render = 1 )
```

Draw this item and its children.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X</td>
</tr>
<tr>
<td>in,out</td>
<td>Y</td>
</tr>
<tr>
<td>in</td>
<td>W</td>
</tr>
<tr>
<td>in</td>
<td>itemfocus</td>
</tr>
<tr>
<td>in,out</td>
<td>tree_item_xmax</td>
</tr>
<tr>
<td>in</td>
<td>lastchild</td>
</tr>
<tr>
<td>in</td>
<td>render</td>
</tr>
</tbody>
</table>

Version

1.3.3 ABI feature: modified parameters

12.153.3.13 draw_horizontal_connector()

```c
void Fl_Tree_Item::draw_horizontal_connector (  
    int x1,  
    int x2,  
    int y,  
    const Fl_Tree_Prefs & prefs ) [protected]
```

Internal: Horizontal connector line based on preference settings.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x1</td>
</tr>
<tr>
<td>in</td>
<td>x2</td>
</tr>
<tr>
<td>in</td>
<td>y</td>
</tr>
<tr>
<td>in</td>
<td>prefs</td>
</tr>
</tbody>
</table>

12.153.3.14 draw_item_content()

```c
int Fl_Tree_Item::draw_item_content (  
    int render ) [virtual]
```

Draw the item content.

This method can be overridden to implement custom drawing by filling the label_[xywh]() area with content.

A minimal example of how to override draw_item_content() and draw just a normal item’s background and label ourselves:

```c
class MyTreeItem : public Fl_Tree_Item {  
public:  
    MyTreeItem(); { }  
    ~MyTreeItem(); } }  
    // DRAW OUR CUSTOM CONTENT FOR THE ITEM
```
int draw_item_content(int render) {  
    // Our item's dimensions + text content  
    int X=label_x(), Y=label_y(), W=label_w(), H=label_h();  
    const char *text = label() ? label() : "";  
    // Rendering? Do any drawing that's needed  
    if ( render ) {  
        // Draw bg -- a filled rectangle  
        fl_color(drawbgcolor()); fl_rectf(X,Y,W,H);  
        // Draw label  
        fl_font(labelfont(), labelsize()); // use item's label font/size  
        fl_color(drawfgcolor()); // use recommended fg color  
        fl_draw(text, X,Y,W,H, FL_ALIGN_LEFT); // draw the item's label  
    }  
    // Rendered or not, we must calculate content's max X position  
    int lw=0, lh=0;  
    fl_measure(text, lw, lh); // get width of label text  
    return X + lw; // return X + label width  
}  

You can draw anything you want inside draw_item_content() using any of the fl_draw.H functions, as long as it's within the label's xywh area.

To add instances of your custom item to the tree, you can use:

// Example #1: using add()  
MyTreeItem *bart = new MyTreeItem(..); // class derived from Fl_Tree_Item  
tree->add("/Simpsons/Bart", bart); // Add item as /Simpsons/Bart  
..or you can insert or replace existing items:  
// Example #2: using replace()  
MyTreeItem *marge = new MyTreeItem(..); // class derived from Fl_Tree_Item  
item = tree->add("/Simpsons/Marge"); // create item  
item->replace(mi); // replace it with our own

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in render</td>
<td>Whether we should render content (1), or just tally the geometry (0). Fl_Tree may want only to find the widest item in the tree for scrollbar calculations.</td>
</tr>
</tbody>
</table>

Returns

the right-most X coordinate, or 'xmax' of content we drew, i.e. the "scrollable" content. The tree uses the largest xmax to determine the maximum width of the tree's content (needed for e.g. computing the horizontal scrollbar's size).

Version

1.3.3 ABI feature

12.153.3.15 draw_vertical_connector()

void Fl_Tree_Item::draw_vertical_connector (  
    int x,  
    int y1,  
    int y2,  
    const Fl_Tree_Prefs & prefs ) [protected]

Internal: Vertical connector line based on preference settings.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in x</td>
<td>The x position of the vertical connector</td>
</tr>
<tr>
<td>in y1</td>
<td>The top of the vertical connector</td>
</tr>
<tr>
<td>in y2</td>
<td>The bottom of the vertical connector</td>
</tr>
<tr>
<td>in prefs</td>
<td>The Fl_Tree prefs</td>
</tr>
</tbody>
</table>
12.153.3.16 drawbgcolor()

`Fl_Color Fl_Tree_Item::drawbgcolor ( ) const` [protected]

Returns the recommended background color used for drawing this item.

See also

draw_item_content()

Version

1.3.3 ABI

12.153.3.17 drawfgcolor()

`Fl_Color Fl_Tree_Item::drawfgcolor ( ) const` [protected]

Returns the recommended foreground color used for drawing this item.

See also

draw_item_content()

Version

1.3.3 ABI ABI

12.153.3.18 find_child() [1/2]

```c
int Fl_Tree_Item::find_child ( const char * name )
```

Return the index of the immediate child of this item that has the label 'name'.

Returns

index of found item, or -1 if not found.

Version

1.3.0 release

12.153.3.19 find_child() [2/2]

```c
int Fl_Tree_Item::find_child ( Fl_Tree_Item * item )
```

Find the index number for the specified 'item' in the current item's list of children.

Returns

the index, or -1 if not found.

12.153.3.20 find_child_item() [1/2]

```c
const Fl_Tree_Item * Fl_Tree_Item::find_child_item ( char ** arr ) const
```

Find child item by descending array 'arr' of names.
Does not include self in search. Only `Fl_Tree` should need this method.

Returns

item, or 0 if not found

Version

1.3.0 release
12.153.3.21 find_child_item() [2/2]

const Fl_Tree_Item * Fl_Tree_Item::find_child_item (const char * name) const

Return the /immediate/ child of current item that has the label 'name'.

Returns

const found item, or 0 if not found.

Version

1.3.3

12.153.3.22 find_clicked()

const Fl_Tree_Item * Fl_Tree_Item::find_clicked (const Fl_Tree_Prefs & prefs, int yonly = 0) const

Find the item that the last event was over.
If 'yonly' is 1, only check event's y value, don't care about x.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>prefs</th>
<th>The parent tree's Fl_Tree_Prefs</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>yonly</td>
<td>– 0: check both event's X and Y values. – 1: only check event's Y value, don't care about X.</td>
</tr>
</tbody>
</table>

Returns

pointer to clicked item, or NULL if none found

Version

1.3.3 ABI feature

12.153.3.23 find_item()

const Fl_Tree_Item * Fl_Tree_Item::find_item (char ** names) const

Find item by descending array of 'names'.
Includes self in search. Only Fl_Tree should need this method. Use Fl_Tree::find_item() instead.

Returns

const item, or 0 if not found

12.153.3.24 hide_widgets()

void Fl_Tree_Item::hide_widgets () [protected]

Internal: Hide the FLTK widget() for this item and all children.
Used by close() to hide widgets.

12.153.3.25 insert()

Fl_Tree_Item * Fl_Tree_Item::insert (const Fl_Tree_Prefs & prefs, const char * new_label, int pos = 0)

Insert a new item named 'new_label' into current item's children at a specified position 'pos'.
If pos is out of range the new item is
• prepended if \( pos < 0 \) or
• appended if \( pos > \text{item->children()} \).

Returns
the new item inserted

See also
Fl_Tree::insert()

12.153.3.26  insert_above()

\texttt{Fl\_Tree\_Item} \* Fl\_Tree\_Item::insert_above (  
\hspace{1em} \text{const Fl\_Tree\_Prefs} \& \text{prefs,}  
\hspace{1em} \text{const char} \* \text{new\_label} )

Insert a new item named \texttt{'new\_label'} above this item.

Returns
the new item inserted, or 0 if an error occurred.

12.153.3.27  label()

\texttt{void Fl\_Tree\_Item::label (}  
\hspace{1em} \text{const char} \* \text{name} )

Set the label to \texttt{'name'}.
Makes and manages an internal copy of \texttt{'name'}.

12.153.3.28  label_h()

\texttt{int Fl\_Tree\_Item::label_h (} \text{const [inline]}  
\hspace{1em} \text{The item's label height.}  
\hspace{1em} \text{Version}  
\hspace{2em} 1.3.3

12.153.3.29  label_w()

\texttt{int Fl\_Tree\_Item::label_w (} \text{const [inline]}  
\hspace{1em} \text{The item's maximum label width to right edge of Fl\_Tree's inner width within scrollbars.}  
\hspace{1em} \text{Version}  
\hspace{2em} 1.3.3

12.153.3.30  label_x()

\texttt{int Fl\_Tree\_Item::label_x (} \text{const [inline]}  
\hspace{1em} \text{The item's label x position relative to the window.}  
\hspace{1em} \text{Version}  
\hspace{2em} 1.3.3
12.153.31  **label_y()**

```cpp
int Fl_Tree_Item::label_y ( ) const [inline]
```

The item’s label y position relative to the window.

**Version**

1.3.3

12.153.32  **labelbgcolor() [1/2]**

```cpp
Fl_Color Fl_Tree_Item::labelbgcolor ( ) const [inline]
```

Return item’s label background text color.

If the color is 0xffffffff, the default behavior is the parent tree’s bg color will be used. (An overloaded `draw_item_content()` can override this behavior.)

12.153.33  **labelbgcolor() [2/2]**

```cpp
void Fl_Tree_Item::labelbgcolor ( Fl_Color val ) [inline]
```

Set item’s label background color.

A special case is made for color 0xffffffff which uses the parent tree's bg color.

12.153.34  **move() [1/2]**

```cpp
int Fl_Tree_Item::move ( Fl_Tree_Item * item, int op = 0, int pos = 0 )
```

Move the current item above/below/into the specified `item`, where `op` determines the type of move:

- 0: move above `item` (`pos` ignored)
- 1: move below `item` (`pos` ignored)
- 2: move into `item` as a child (at optional position `pos`)

..and `pos` determines an optional index position after the move.

**Returns**

0 on success. a negative number on error:

-1: one of the items has no parent
-2: item’s index could not be determined
-3: bad ‘op’
-4: index range error
-5: could not deparent
-6: could not reparent at ‘pos’

(Other return values reserved for future use.)

**See also**

`move_above(), move_below(), move_into(), move(int,int)`

Generated by Doxygen
**12.153.3.35 move() [2/2]**

```cpp
int Fl_Tree_Item::move (int to, int from)
```

Move an item within its parent using index numbers.
Item is moved 'to' its new position 'from' its old position.

Returns

- 0: Success
- -1: range error (e.g. if 'to' or 'from' out of range).
- (Other return values reserved for future use)

See also

move_above(), move_below(), move_into(), move(Fl_Tree_Item*,int,int)

**12.153.3.36 move_above()**

```cpp
int Fl_Tree_Item::move_above (Fl_Tree_Item* item)
```

Move the current item above the specified 'item'.
This is the equivalent of calling move(item,0,0).

Returns

- 0 on success.
- On error returns a negative value; see move(Fl_Tree_Item*,int,int) for possible error codes.

See also

move_below(), move_into(), move(int,int), move(Fl_Tree_Item*,int,int)

**12.153.3.37 move_below()**

```cpp
int Fl_Tree_Item::move_below (Fl_Tree_Item* item)
```

Move the current item below the specified 'item'.
This is the equivalent of calling move(item,1,0).

Returns

- 0 on success.
- On error returns a negative value; see move(Fl_Tree_Item*,int,int) for possible error codes.

See also

move_above(), move_into(), move(int,int), move(Fl_Tree_Item*,int,int)

**12.153.3.38 move_into()**

```cpp
int Fl_Tree_Item::move_into (Fl_Tree_Item* item, int pos = 0)
```

Parent the current item as a child of the specified 'item'.
This is the equivalent of calling move(item,2,pos).
Returns

0 on success.
On error returns a negative value; see `move(Fl_Tree_Item*,int,int)` for possible error codes.

See also

`move_above(), move_below(), move(int,int), move(Fl_Tree_Item*,int,int)`

12.153.3.39 next()

`Fl_Tree_Item* Fl_Tree_Item::next()`

Return the next item in the tree.
This method can be used to walk the tree forward. For an example of how to use this method, see `Fl_Tree::first()`.

Returns

the next item in the tree, or 0 if there's no more items.

12.153.3.40 next_displayed()

`Fl_Tree_Item* Fl_Tree_Item::next_displayed(Fl_Tree_Prefs & prefs)`

Same as `next_visible()`.

Deprecated in 1.3.3 for confusing name, use `next_visible()` instead

12.153.3.41 next_sibling()

`Fl_Tree_Item* Fl_Tree_Item::next_sibling()`

Return this item's next sibling.
Moves to the next item below us at the same level (sibling). Use this to move down the tree without changing `depth()`, effectively skipping over this item's children/descendents.

Returns

item's next sibling, or 0 if none.

12.153.3.42 next_visible()

`Fl_Tree_Item* Fl_Tree_Item::next_visible(Fl_Tree_Prefs & prefs)`

Return the next open(), visible() item.
(If this item has children and is closed, children are skipped)
This method can be used to walk the tree forward, skipping items that are not currently open/visible to the user.

Returns

the next open() visible() item below us, or 0 if there's no more items.

Version

1.3.3

12.153.3.43 parent()

`void Fl_Tree_Item::parent(Fl_Tree_Item* val)`

Set the parent for this item.
Should only be used by `Fl_Tree`'s internals.
12.153.3.44  prefs()

const Fl_Tree_Prefs & Fl_Tree_Item::prefs ( ) const
Return the parent tree's prefs.

Returns
    a reference to the parent tree's Fl_Tree_Prefs

Version
    1.3.3 ABI feature

12.153.3.45  prev()

Fl_Tree_Item * Fl_Tree_Item::prev ( )
Return the previous item in the tree.
This method can be used to walk the tree backwards. For an example of how to use this method, see Fl_Tree::last().

Returns
    the previous item in the tree, or 0 if there's no item above this one (hit the root).

12.153.3.46  prev_displayed()

Fl_Tree_Item * Fl_Tree_Item::prev_displayed ( Fl_Tree_Prefs & prefs )
Same as prev_visible().

Deprecated in 1.3.3 for confusing name, use prev_visible()

12.153.3.47  prev_sibling()

Fl_Tree_Item * Fl_Tree_Item::prev_sibling ( )
Return this item's previous sibling.
Moves to the previous item above us at the same level (sibling). Use this to move up the tree without changing depth().

Returns
    This item's previous sibling, or 0 if none.

12.153.3.48  prev_visible()

Fl_Tree_Item * Fl_Tree_Item::prev_visible ( Fl_Tree_Prefs & prefs )
Return the previous open(), visible() item.
(If this item above us has children and is closed, its children are skipped)
This method can be used to walk the tree backward, skipping items that are not currently open/visible to the user.

Returns
    the previous open() visible() item above us, or 0 if there's no more items.

12.153.3.49  recalc_tree()

void Fl_Tree_Item::recalc_tree ( ) [protected]
Call this when our geometry is changed.
(Font size, label contents, etc) Schedules tree to recalculate itself, as changes to us may affect tree widget's scrollbar visibility and tab sizes.

Version
    1.3.3 ABI
12.153.3.50  remove_child() [1/2]

int Fl_Tree_Item::remove_child ( const char * name )
Remove immediate child (and its children) by its label 'name'.
If more than one item matches 'name', only the first matching item is removed.

Parameters

| in name | The label name of the immediate child to remove |

Returns

0 if removed, -1 if not found.

Version

1.3.3

12.153.3.51  remove_child() [2/2]

int Fl_Tree_Item::remove_child ( Fl_Tree_Item * item )
Remove 'item' from the current item's children.

Returns

0 if removed, -1 if item not an immediate child.

12.153.3.52  reparent()

int Fl_Tree_Item::reparent ( Fl_Tree_Item * newchild, int pos )
Reparent specified item as a child of ourself at position 'pos'.
Typically 'newchild' was recently orphaned with deparent().

Returns

- 0: on success
- -1: on error (e.g. if 'pos' out of range) with no changes made.

See also

deparent()

12.153.3.53  replace()

Fl_Tree_Item * Fl_Tree_Item::replace ( Fl_Tree_Item * newitem )
Replace the current item with a new item.
The current item is destroyed if successful. No checks are made to see if an item with the same name exists.
This method can be used to, for example, install 'custom' items into the tree derived from Fl_Tree_Item; see draw_item_content().

Parameters

| in newitem | The new item to replace the current item |
Returns

newitem on success, NULL if could not be replaced.

See also

Fl_Tree_Item::draw_item_content(), Fl_Tree::root(Fl_Tree_Item*)

Version

1.3.3 ABI feature

**12.153.3.54 replace_child()**

```c
Fl_Tree_Item* Fl_Tree_Item::replace_child (  
    Fl_Tree_Item* olditem,  
    Fl_Tree_Item* newitem)
```

Replace existing child 'olditem' with 'newitem'.
The 'olditem' is destroyed if successful. Can be used to put custom items (derived from Fl_Tree_Item) into the tree. No checks are made to see if an item with the same name exists.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>olditem</th>
<th>The item to be found and replaced</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>newitem</td>
<td>The new item to take the place of 'olditem'</td>
</tr>
</tbody>
</table>

Returns

newitem on success and 'olditem' is destroyed. NULL on error if 'olditem' was not found as an immediate child.

See also

replace(), Fl_Tree_Item::draw()

Version

1.3.3 ABI feature

**12.153.3.55 select()**

```c
void Fl_Tree_Item::select (  
    int val = 1) [inline]
```

Change the item's selection state to the optionally specified 'val'.
If 'val' is not specified, the item will be selected.

**12.153.3.56 select_all()**

```c
int Fl_Tree_Item::select_all () [inline]
```

Select item and all its children.
Returns count of how many items were in the 'deselected' state, ie. how many items were "changed".

**12.153.3.57 show_self()**

```c
void Fl_Tree_Item::show_self (  
    const char* indent = "") const
```

Print the tree as 'ascii art' to stdout.
Used mainly for debugging.
show_widgets()

```c
void Fl_Tree_Item::show_widgets ( ) [protected]
```

Internal: Show the FLTK widget() for this item and all children.
Used by open() to re-show widgets that were hidden by a previous close()

swap_children() [1/2]

```c
int Fl_Tree_Item::swap_children ( Fl_Tree_Item * a, Fl_Tree_Item * b )
```

Swap two of our immediate children, given item pointers.
Use e.g. for sorting.
This method is SLOW because it involves linear lookups.
For speed, use swap_children(int,int) instead.

Parameters

| in | a,b | The item ptrs of the two items to swap. Both must be immediate children of the current item. |

Returns

- 0 : OK
- -1 : failed: item 'a' or 'b' is not our child.

swap_children() [2/2]

```c
void Fl_Tree_Item::swap_children ( int ax, int bx )
```

Swap two of our children, given two child index values 'ax' and 'bx'.
Use e.g. for sorting.
No range checking is done on either index value.

Parameters

| in | ax,bx | the index of the items to swap |

tree() [1/2]

```c
Fl_Tree * Fl_Tree_Item::tree ( ) [inline]
```

Return the tree for this item.

Version

1.3.4

tree() [2/2]

```c
const Fl_Tree * Fl_Tree_Item::tree ( ) const [inline]
```

Return the tree for this item.

Version

1.3.3
12.153.3.63  update_prev_next()

void Fl_Tree_Item::update_prev_next (  
    int index  
)  

Update our _prev_sibling and _next_sibling pointers to point to neighbors given index as being our current position in the parent's item array.  
Call this whenever items in the array are added/removed/moved/swapped/etc.  

Parameters  

| in  | index | Our index# in the parent.  
|-----|-------| Special case if index=-1: become an orphan; null out all parent/sibling associations. |

12.153.3.64  userdeicon() [1/2]

Fl_Image * Fl_Tree_Item::userdeicon () const [inline]  
Return the deactivated version of the user icon, if any.  
Returns 0 if none.  

12.153.3.65  userdeicon() [2/2]

void Fl_Tree_Item::userdeicon (  
    Fl_Image * val  
) [inline]  
Set the usericon to draw when the item is deactivated.  
Use '0' to disable. No internal copy is made; caller must manage icon's memory.  
To create a typical 'grayed out' version of your usericon image, you can do the following:  
// Create tree + usericon for items  
Fl_Tree *tree = new Fl_Tree(...);  
Fl_Image *usr_icon = new Fl_Pixmap(...); // your usericon  
Fl_Image *de_icon = usr_icon->copy(); // make a copy, and..  
de_icon->inactive(); // make it 'grayed out'  
...  
for ( .. ) { // item loop..  
    item = tree->add("..."); // create new item  
    item->usericon(usr_icon); // assign usericon to items  
    item->userdeicon(de_icon); // assign userdeicon to items  
    ..  
}  

In the above example, the app should 'delete' the two icons when they're no longer needed (e.g. after the tree is destroyed)  
Version  

1.3.4  

12.153.3.66  usericon()

void Fl_Tree_Item::usericon (  
    Fl_Image * val  
) [inline]  
Set the item's user icon to an Fl_Image.  
Use '0' to disable. No internal copy is made, caller must manage icon's memory.  
Note, if you expect your items to be deactivated(), use userdeicon(Fl_Image*) to set up a 'grayed out' version of your icon to be used for display.  
See also  
userdeicon(Fl_Image*)  

12.153.3.67  visible_r()

int Fl_Tree_Item::visible_r ( ) const  
See if item and all its parents are open() and visible().
Returns

1 – item and its parents are open() and visible() 0 – item (or one of its parents) are invisible or closed.

The documentation for this class was generated from the following files:

- Fl_Tree_Item.H
- Fl_Tree_Item.cxx

### 12.154 Fl_Tree_Item_Array Class Reference

Manages an array of Fl_Tree_Item pointers.

```c
#include <Fl_Tree_Item_Array.H>
```

#### Public Member Functions

- **void** add (Fl_Tree_Item *val)
  - Add an item to the end of the array.
- **void** clear ()
  - Clear the entire array.
- **int** deparent (int pos)
  - Deparent item at 'pos' from our list of children.
- **Fl_Tree_Item_Array** (const Fl_Tree_Item_Array *o)
  - Copy constructor. Makes new copy of array, with new instances of each item.
- **Fl_Tree_Item_Array** (int new_chunksize=10)
  - Constructor; creates an empty array.
- **void** insert (int pos, Fl_Tree_Item *new_item)
  - Insert an item at index position pos.
- **int** manage_item_destroy () const
- **void** manage_item_destroy (int val)
  - Option to control if Fl_Tree_Item_Array's destructor will also destroy the Fl_Tree_Item's.
- **int** move (int to, int from)
  - Move item at 'from' to new position 'to' in the array.
- **Fl_Tree_Item * operator[] (int i)**
  - Return the item and index i.
- **const Fl_Tree_Item * operator[] (int i) const**
  - Const version of operator[](int i)
- **int** remove (Fl_Tree_Item *item)
  - Remove the item from the array.
- **void** remove (int index)
  - Remove the item at.
- **int** reparent (Fl_Tree_Item *item, Fl_Tree_Item *newparent, int pos)
  - Reparent specified item as a child of ourself.
- **void** replace (int pos, Fl_Tree_Item *new_item)
  - Replace the item at index with newitem.
- **void** swap (int ax, int bx)
  - Swap the two items at index positions ax and bx.
- **int** total () const
  - Return the total items in the array, or 0 if empty.
- **~Fl_Tree_Item_Array ()**
  - Destructor. Calls each item's destructor, destroys internal_items array.
12.154.1 Detailed Description

Manages an array of Fl_Tree_Item pointers. Because FLTK 1.x.x. has mandated that templates and STL not be used, we use this class to dynamically manage the arrays. None of the methods do range checking on index values; the caller must be sure that index values are within the range 0<index<total() (unless otherwise noted).

12.154.2 Constructor & Destructor Documentation

12.154.2.1 Fl_Tree_Item_Array()

Fl_Tree_Item_Array::Fl_Tree_Item_Array ( 
    int new_chunksize = 10 )

Constructor; creates an empty array. The optional ‘chunksize’ can be specified to optimize memory allocation for potentially large arrays. Default chunksize is 10.

12.154.3 Member Function Documentation

12.154.3.1 add()

void Fl_Tree_Item_Array::add ( 
    Fl_Tree_Item * val )

Add an item+ to the end of the array. Assumes the item was created with ‘new’, and will remain allocated. Fl_Tree_Item_Array will handle calling the item’s destructor when the array is cleared or the item remove()'ed.

12.154.3.2 clear()

void Fl_Tree_Item_Array::clear ( )

Clear the entire array. Each item will be deleted (destructors will be called), and the array will be cleared. total() will return 0.

12.154.3.3 deparent()

int Fl_Tree_Item_Array::deparent ( 
    int pos )

Deparent item at 'pos' from our list of children. Similar to a remove() without the destruction of the item. This creates an orphaned item (still allocated, has no parent) which soon after is typically reparented elsewhere. \returns 0 on success, -1 on error (e.g. if \p 'pos' out of range)

12.154.3.4 insert()

void Fl_Tree_Item_Array::insert ( 
    int pos, 
    Fl_Tree_Item * new_item )

Insert an item at index position pos. Handles enlarging array if needed, total increased by 1. If \p pos \>= total(), the item is appended to the array. If \p pos \< 0, the item is prepended (works like pos == 0).
12.154.3.5  **manage_item_destroy()**

```c
void Fl_Tree_Item_Array::manage_item_destroy (  
    int val  )  [inline]
```

Option to control if `Fl_Tree_Item_Array`'s destructor will also destroy the `Fl_Tree_Item`'s.
If set: items and item array is destroyed. If clear: only the item array is destroyed, not items themselves.

12.154.3.6  **move()**

```c
int Fl_Tree_Item_Array::move (  
    int to,  
    int from )
```

Move item at 'from' to new position 'to' in the array.
Due to how the moving an item shuffles the array around, a positional 'move' implies things that may not be obvious:

- When 'from' moved lower in tree, appears BELOW item that was at 'to'.
- When 'from' moved higher in tree, appears ABOVE item that was at 'to'.

Returns
0 on success, -1 on range error (e.g. if 'to' or 'from' out of range)

12.154.3.7  **remove() [1/2]**

```c
int Fl_Tree_Item_Array::remove (  
    Fl_Tree_Item * item)
```

Remove the item from the array.
\returns 0 if removed, or -1 if the item was not in the array.

12.154.3.8  **remove() [2/2]**

```c
void Fl_Tree_Item_Array::remove (  
    int index )
```

Remove the item at.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>index</th>
<th>from the array.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>The item will be delete'd (if non-NULL), so its destructor will be called.</td>
</tr>
</tbody>
</table>

12.154.3.9  **reparent()**

```c
int Fl_Tree_Item_Array::reparent (  
    Fl_Tree_Item * item,  
    Fl_Tree_Item * newparent,  
    int pos )
```

Reparent specified item as a child of ourself.
Typically 'newchild' was recently orphaned with deparent().
\returns 0 on success, -1 on error (e.g. if \p 'pos' out of range)

12.154.3.10  **replace()**

```c
void Fl_Tree_Item_Array::replace (  
```


Replace the item at index with newitem.
Old item at index position will be destroyed, and the new item will take it's place, and stitched into the linked list.

The documentation for this class was generated from the following files:

- Fl_Tree_Item_Array.H
- Fl_Tree_Item_Array.cxx

### 12.155 Fl_Tree_Prefs Class Reference

Tree widget's preferences.
#include <Fl_Tree_Prefs.H>

#### Public Member Functions

- **Fl_Image * closedeicon () const**
  
  Return the deactivated version of the close icon, if any.

- **Fl_Image * closeicon () const**
  
  Gets the default 'close' icon Returns the Fl_Image+ of the icon, or 0 if none.

- **void closeicon (Fl_Image +val)**

  Sets the icon to be used as the 'close' icon.

- **Fl_Color connectorcolor () const**

  Get the connector color used for tree connection lines.

- **void connectorcolor (Fl_Color val)**

  Set the connector color used for tree connection lines.

- **Fl_Tree_Connector connectorstyle () const**

  Get the connector style.

- **void connectorstyle (Fl_Tree_Connector val)**

  Set the connector style.

- **void connectorstyle (int val)**

  Set the connector style [integer].

- **int connectorwidth () const**

  Get the tree connection line's width.

- **void connectorwidth (int val)**

  Set the tree connection line's width.

- **void do_item_draw_callback (Fl_Tree_Item +o) const**

- **Fl_Tree_Prefs ()**

  Fl_Tree_Prefs constructor.

- **Fl_Tree_Item_Draw_Callback * item_draw_callback () const**

- **void item_draw_callback (Fl_Tree_Item_Draw_Callback +cb, void +data=0)**

- **Fl_Tree_Item_Draw_Mode item_draw_mode () const**

  Get the 'item draw mode' used for the tree.

- **void item_draw_mode (Fl_Tree_Item_Draw_Mode val)**

  Set the 'item draw mode' used for the tree to val.

- **void * item_draw_user_data () const**

- **Fl_Color item_labelbgcolor () const**

  Get the default label background color.

- **void item_labelbgcolor (Fl_Color val)**

  Set the default label background color.

- **Fl_Color item_labelsfgcolor () const**

  Get the default label foreground color.

- **void item_labelsfgcolor (Fl_Color val)**
Set the default label foreground color.

- FL_Font item_labelfont () const
  Return the label's font.

- void item_labelfont (FL_Font val)
  Set the label's font to val.

- FL_Fontsize item_labelsize () const
  Return the label's size in pixels.

- void item_labelsize (FL_Fontsize val)
  Set the label's size in pixels to val.

- FL_Tree_Item_Reselect_Mode item_reselect_mode () const
  Returns the current item re/selection mode.

- void item_reselect_mode (FL_Tree_Item_Reselect_Mode mode)
  Sets the item re/selection mode.

- FL_Color labelbgcolor () const
  Obsolete: Get the default label background color. Please use item_labelbgcolor() instead.

- void labelbgcolor (FL_Color val)
  Obsolete: Set the default label background color. Please use item_labelbgcolor(FL_Color) instead.

- FL_Color labelfgcolor () const
  Obsolete: Get the default label foreground color. Please use item_labelfgcolor() instead.

- void labelfgcolor (FL_Color val)
  Obsolete: Set the default label foreground color. Please use item_labelfgcolor(FL_Color) instead.

- FL_Font labelfont () const
  Obsolete: Return the label's font. Please use item_labelfont() instead.

- void labelfont (FL_Font val)
  Obsolete: Set the label's font to val. Please use item_labelfont(FL_Font) instead.

- int labelmarginleft () const
  Get the label's left margin value in pixels.

- void labelmarginleft (int val)
  Set the label's left margin value in pixels.

- FL_Fontsize labelsize () const
  Obsolete: Return the label's size in pixels. Please use item_labelsize() instead.

- void labelsize (FL_Fontsize val)
  Obsolete: Set the label's size in pixels to val. Please use item_labelsize(FL_Fontsize) instead.

- int linespacing () const
  Get the line spacing value in pixels.

- void linespacing (int val)
  Set the line spacing value in pixels.

- int marginbottom () const
  Get the bottom margin's value in pixels.

- void marginbottom (int val)
  Set the bottom margin's value in pixels This is the extra distance the vertical scroller lets you travel.

- int marginleft () const
  Get the left margin's value in pixels.

- void marginleft (int val)
  Set the left margin's value in pixels.

- int margintop () const
  Get the top margin's value in pixels.

- void margintop (int val)
  Set the top margin's value in pixels.

- int openchild_marginbottom () const
  Get the margin below an open child in pixels.
1820 Class Documentation

- **void openchild_marginbottom**(int val)
  
  Set the margin below an open child in pixels.

- **Fl管局Its* opendeicon () const
  
  Return the deactivated version of the open icon, if any.

- **Fl管局Its* openicon () const
  
  Get the current default 'open' icon.

- **void openicon (Fl管局Its* val)
  
  Sets the default icon to be used as the 'open' icon when items are add()ed to the tree.

- **Fl管局Its* selectbox () const
  
  Get the current default 'open' icon.

- **void selectbox (Fl管局Its* val)
  
  Set the default selection box's box drawing style to val.

- **Fl管局Its* selectmode () const
  
  Get the selection mode used for the tree.

- **void selectmode (Fl管局Its* val)
  
  Set the selection mode used for the tree to val.

- **char showcollapse () const
  
  Returns 1 if the collapse icon is enabled, 0 if not.

- **void showcollapse (int val)
  
  Set if we should show the collapse icon or not.

- **int showroot () const
  
  Returns 1 if the root item is to be shown, or 0 if not.

- **void showroot (int val)
  
  Set if the root item should be shown or not.

- **Fl管局Its* sortorder () const
  
  Get the default sort order value.

- **void sortorder (Fl管局Its* val)
  
  Set the default sort order value.

- **Fl管局Its* userdeicon () const
  
  Return the deactivated version of the user icon, if any.

- **Fl管局Its* usericon () const
  
  Gets the default 'user icon' (default is 0)

- **void usericon (Fl管局Its* val)
  
  Sets the default 'user icon' Returns the Fl管局Its* of the icon, or 0 if none (default).

- **int usericonmarginleft () const
  
  Get the user icon's left margin value in pixels.

- **void usericonmarginleft (int val)
  
  Set the user icon's left margin value in pixels.

- **int widgetmarginleft () const
  
  Get the widget's left margin value in pixels.

- **void widgetmarginleft (int val)
  
  Set the widget's left margin value in pixels.

- **~Fl管局Its* ()
  
  Fl管局Its destructor.

12.155.1 Detailed Description

Tree widget's preferences.  
Fl管局Its Preferences class.
This class manages the Fl管局Its's defaults. You should probably be using the methods in Fl管局Its instead of trying to accessing tree's preferences settings directly.
12.155.2 Member Function Documentation

12.155.2.1 closedeicon()

Fl_Image * Fl_Tree_Prefs::closedeicon ( ) const [inline]
Return the deactivated version of the close icon, if any.
Returns 0 if none.

12.155.2.2 closeicon()

void Fl_Tree_Prefs::closeicon ( Fl_Image * val )
Sets the icon to be used as the 'close' icon.
This overrides the built in default '[-]' icon.

Parameters

| in  | val | - The new image, or zero to use the default [-] icon. |

12.155.2.3 item_draw_mode()

void Fl_Tree_Prefs::item_draw_mode ( Fl_Tree_Item_Draw_Mode val ) [inline]
Set the 'item draw mode' used for the tree to val.
This affects how items in the tree are drawn, such as when a widget() is defined. See Fl_Tree_Item_Draw_Mode for possible values.

12.155.2.4 item_labelbgcolor() [1/2]

Fl_Color Fl_Tree_Prefs::item_labelbgcolor ( ) const [inline]
Get the default label background color.
This returns the Fl_Tree::color() unless item_labelbgcolor() has been set explicitly.

12.155.2.5 item_labelbgcolor() [2/2]

void Fl_Tree_Prefs::item_labelbgcolor ( Fl_Color val ) [inline]
Set the default label background color.
Once set, overrides the default behavior of using Fl_Tree::color().

12.155.2.6 marginbottom()

int Fl_Tree_Prefs::marginbottom ( ) const [inline]
Get the bottom margin's value in pixels.
This is the extra distance the vertical scroller lets you travel.

12.155.2.7 opendeicon()

Fl_Image * Fl_Tree_Prefs::opendeicon ( ) const [inline]
Return the deactivated version of the open icon, if any.
Returns 0 if none.

12.155.2.8 openicon() [1/2]

Fl_Image * Fl_Tree_Prefs::openicon ( ) const [inline]
Get the current default 'open' icon.
Returns the Fl_Image* of the icon, or 0 if none.
12.155.2.9  openicon() [2/2]

void Fl_Tree_Prefs::openicon (  
    Fl_Image * val )
Sets the default icon to be used as the 'open' icon when items are add()ed to the tree.  
This overrides the built in default '[+]' icon.

Parameters

| in | val | – The new image, or zero to use the default [+ ] icon. |

12.155.2.10  selectmode()

void Fl_Tree_Prefs::selectmode (  
    Fl_Tree_Select val ) [inline]
Set the selection mode used for the tree to val.  
This affects how items in the tree are selected when clicked on and dragged over by the mouse.  
See Fl_Tree_Select for possible values.

12.155.2.11  showcollapse()

void Fl_Tree_Prefs::showcollapse (  
    int val ) [inline]
Set if we should show the collapse icon or not.  
If collapse icons are disabled, the user will not be able to interactively collapse items in the tree, unless the application provides some other means via open() and close().

Parameters

| in | val | 1: shows collapse icons (default), 0: hides collapse icons. |

12.155.2.12  showroot()

void Fl_Tree_Prefs::showroot (  
    int val ) [inline]
Set if the root item should be shown or not.

Parameters

| in | val | 1 – show the root item (default)  

0 – hide the root item. |

12.155.2.13  sortorder()

void Fl_Tree_Prefs::sortorder (  
    Fl_Tree_Sort val ) [inline]
Set the default sort order value.  
Defines the order new items appear when add()ed to the tree.  
See Fl_Tree_Sort for possible values.

12.155.2.14  userdeicon()

Fl_Image * Fl_Tree_Prefs::userdeicon ( ) const [inline]
Return the deactivated version of the user icon, if any.  
Returns 0 if none.

The documentation for this class was generated from the following files:
The **Fl_Valuator** class controls a single floating-point value and provides a consistent interface to set the value, range, and step, and insures that callbacks are done the same for every object.

```cpp
#include <Fl_Valuator.H>
```

Inheritance diagram for **Fl_Valuator**:

```
Fl_Widget
  └── Fl_Valuator
      ├── Fl_Adjuster
      ├── Fl_Counter
      ├── Fl_Dial
      │    └── Fl_Fill_Dial
      │         └── Fl_Line_Dial
      ├── Fl_Roller
      └── Fl_Slider
          └── Fl_Scrollbar
```

### Public Member Functions

- **void bounds** (double a, double b)
  
  *Sets the minimum (a) and maximum (b) values for the valuator widget.*

- **double clamp** (double)
  
  *Clamps the passed value to the valuator range.*

- **virtual int format** (char *)
  
  *Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.*

- **double increment** (double, int)
  
  *Adds n times the step value to the passed value.*

- **double maximum** () const
  
  *Gets the maximum value for the valuator.*

- **void maximum** (double a)
  
  *Sets the maximum value for the valuator.*

- **double minimum** () const
  
  *Gets the minimum value for the valuator.*

- **void minimum** (double a)
  
  *Sets the minimum value for the valuator.*

- **void precision** (int digits)
  
  *Sets the step value to 1.0 / 10^digits.*

- **void range** (double a, double b)
  
  *Sets the minimum and maximum values for the valuator.*

- **double round** (double)
  
  *Round the passed value to the nearest step increment.*

- **double step** () const
  
  *Gets or sets the step value.*

- **void step** (double a, int b)
See double Fl_Valuator::step() const

• void step (double s)
  See double Fl_Valuator::step() const.
• void step (int a)
  See double Fl_Valuator::step() const

• double value () const
  Gets the floating point(double) value.
• int value (double)
  Sets the current value.

Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
  Activates the widget.
• unsigned int active () const
  Returns whether the widget is active.
• int active_r () const
  Returns whether the widget and all of its parents are active.
• Fl_Align align () const
  Gets the label alignment.
• void align (Fl_Align alignment)
  Sets the label alignment.
• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
• virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
• virtual class Fl_Gl_Window const ∗ as_gl_window () const
• virtual Fl_Group ∗ as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• virtual Fl_Group const ∗ as_group () const
• virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• virtual Fl_Window const ∗ as_window () const
• void bind_deimage (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the inactive state.
• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
• void bind_image (Fl_Image ∗img)
  Sets the image to use as part of the widget label when in the active state.
• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
• Fl_Boxtype box () const
  Gets the box type of the widget.
• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.
• Fl_Callback_p callback () const
 Gets the current callback function for the widget.
• void callback (Fl_Callback ∗cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback ∗cb, void ∗p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 ∗cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 ∗cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
  Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget ∗w) const
  Checks if w is a child of this widget.
• void copy_label (const char ∗new_label)
  Sets the current label.
• void copy_tooltip (const char ∗text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
- `Fl_Image * deimage ()`
  Gets the image that is used as part of the widget label when in the inactive state.
- `const Fl_Image * deimage () const`
  Gets the image that is used as part of the widget label when in the inactive state.
- `void deimage (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the inactive state.
- `void deimage (Fl_Image *img)`
  Sets the image to use as part of the widget label when in the inactive state.
- `int deimage_bound () const`
  Returns whether the inactive image is managed by the widget.
- `void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with default arguments.
- `void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.
- `void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.
- `virtual void draw ()=0`
  Draws the widget.
- `void draw_label (int, int, int, int, Fl_Align) const`
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- `int h () const`
  Gets the widget height.
- `virtual int handle (int event)`
  Handles the specified event.
- `virtual void hide ()`
  Makes a widget invisible.
- `Fl_Image * image ()`
  Gets the image that is used as part of the widget label when in the active state.
- `const Fl_Image * image () const`
  Gets the image that is used as part of the widget label when in the active state.
- `void image (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the active state.
- `void image (Fl_Image *img)`
  Sets the image to use as part of the widget label when in the active state.
- `int image_bound () const`
  Returns whether the image is managed by the widget.
- `int inside (const Fl_Widget *wgt) const`
  Checks if this widget is a child of `wgt`.
- `int is_label_copied () const`
  Returns whether the current label was assigned with `copy_label()`.
- `const char * label () const`
  Gets the current label text.
- `void label (const char *text)`
  Sets the current label pointer.
- `void label (Fl_Labeltype a, const char *b)`
  Shortcut to set the label text and type in one call.
- `Fl_Color labelcolor () const`
  Gets the label color.
- `void labelcolor (Fl_Color c)`
  Sets the label color.
- `Fl_Font labelfont () const`
Gets the font to use.

- void labelfont (Fl_Font f)
  Sets the font to use.

- Fl_Fontsize labelsizes () const
  Gets the font size in pixels.

- void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.

- Fl_Labeltype labeltype () const
  Gets the label type.

- void labeltype (Fl_Labeltype a)
  Sets the label type.

- void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.

- bool needs_keyboard () const
  Returns whether this widget needs a keyboard.

- void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.

- unsigned int output () const
  Returns if a widget is used for output only.

- Fl_Group * parent () const
  Returns a pointer to the parent widget.

- void parent (Fl_Group *p)
  Internal use only - "for hacks only".

- void position (int X, int Y)
  Repositions the window or widget.

- void redraw ()
  Schedules the drawing of the widget.

- void redraw_label ()
  Schedules the drawing of the label.

- virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.

- Fl_Color selection_color () const
  Gets the selection color.

- void selection_color (Fl_Color a)
  Sets the selection color.

- void set_active ()
  Marks the widget as active without sending events or changing focus.

- void set_changed ()
  Marks the value of the widget as changed.

- void set_output ()
  Sets a widget to output only.

- void set_visible ()
  Makes the widget visible.

- void set_visible_focus ()
  Enables keyboard focus navigation with this widget.

- int shortcut_label () const
  Returns whether the widget’s label uses ’&’ to indicate shortcuts.

- void shortcut_label (int value)
  Sets whether the widget’s label uses ’&’ to indicate shortcuts.

- virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int testShortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.
• const char * tooltip () const
  Gets the current tooltip text.
• void tooltip (const char *text)
  Sets the current tooltip text.
• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
  Gets the widget type.
• void type (uchar t)
  Sets the widget type.
• int useAccentsMenu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * user_data () const
  Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.
• void user_data (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  Returns whether a widget is visible.
• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.
• void visible_focus (int v)
  Modifies keyboard focus navigation.
• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl_When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl_Widget ()
  Destroys the widget.
Protected Member Functions

- **Fl_Valuator** (int X, int Y, int W, int H, const char *L)
  Creates a new Fl_Valuator widget using the given position, size, and label string.

- **void handle_drag** (double newvalue)
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.

- **void handle_push**()
  Stores the current value in the previous value.

- **void handle_release**()
  Called after an FL_WHEN_RELEASE event is received and before the callback.

- **int horizontal**() const
  Tells if the valuator is an FL_HORIZONTAL one.

- **double previous_value**() const
  Gets the previous floating point value before an event changed it.

- **void set_value** (double v)
  Sets the current floating point value.

- **double softclamp** (double)
  Clamps the value, but accepts v if the previous value is not already out of range.

- **virtual void value_damage**()
  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- **void clear_flag** (unsigned int c)
  Clears a flag in the flags mask.

- **void draw_backdrop** () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- **void draw_box** () const
  Draws the widget box according its box style.

- **void draw_box** (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

- **void draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- **void draw_focus** () const
  Draws a focus rectangle around the widget.

- **void draw_focus** (Fl_Boxtype t, int x, int y, int w, int h)
  Draws a focus rectangle around the widget.

- **void draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg)
  Draws a focus box for the widget at the given position and size.

- **void draw_label** () const
  Draws the widget's label at the defined label position.

- **void draw_label** (int, int, int, int) const
  Draws the label in an arbitrary bounding box.

- **Fl_Widget** (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.

- **unsigned int flags** () const
  Gets the widget flags mask.

- **void h** (int v)
  Internal use only.

- **void set_flag** (unsigned int c)
  Sets a flag in the flags mask.

- **void w** (int v)

Generated by Doxygen
Internal use only.

• void \texttt{x} (int \texttt{v})

Internal use only.

• void \texttt{y} (int \texttt{v})

Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from \texttt{Fl_Widget}

• static void \texttt{default\_callback} (\texttt{Fl\_Widget \ast \texttt{widget}}, void \texttt{\ast \texttt{data}})

\textit{The default callback for all widgets that don't set a callback.}

• static unsigned int \texttt{label\_shortcut} (const char \texttt{\ast \texttt{t}})

\textit{Returns the Unicode value of the \textquotesingle\&x\textquotesingle\ shortcut in a given text.}

• static int \texttt{test\_shortcut} (const char \texttt{\ast \texttt{t}}, const bool require\_alt=false)

\textit{Returns true if the given text \texttt{t} contains the entered \textquotesingle\&x\textquotesingle\ shortcut.}

Protected Types inherited from \texttt{Fl\_Widget}

• enum {

\texttt{INACTIVE = 1 < 0}, \texttt{INVISIBLE = 1 < 1}, \texttt{OUTPUT = 1 < 2}, \texttt{NOBORDER = 1 < 3},
\texttt{FORCE\_POSITION = 1 < 4}, \texttt{NON\_MODAL = 1 < 5}, \texttt{SHORTCUT\_LABEL = 1 < 6}, \texttt{CHANGED = 1 < 7},
\texttt{OVERRIDE = 1 < 8}, \texttt{VISIBLE\_FOCUS = 1 < 9}, \texttt{COPIED\_LABEL = 1 < 10}, \texttt{CLIP\_CHILDREN = 1 < 11},
\texttt{MENU\_WINDOW = 1 < 12}, \texttt{TOOLTIP\_WINDOW = 1 < 13}, \texttt{MODAL = 1 < 14}, \texttt{NO\_OVERLAY = 1 < 15},
\texttt{GROUP\_RELATIVE = 1 < 16}, \texttt{COPIED\_TOOLTIP = 1 < 17}, \texttt{FULLSCREEN = 1 < 18}, \texttt{MAC\_USE\_ACCENTS\_MENU = 1 < 19},
\texttt{NEEDS\_KEYBOARD = 1 < 20}, \texttt{IMAGE\_BOUND = 1 < 21}, \texttt{DEIMAGE\_BOUND = 1 < 22},
\texttt{AUTO\_DELETE\_USER\_DATA = 1 < 23},
\texttt{MAXIMIZED = 1 < 24}, \texttt{USER\_FLAG3 = 1 < 29}, \texttt{USER\_FLAG2 = 1 < 30}, \texttt{USER\_FLAG1 = 1 < 31}}

\textit{flags possible values enumeration.}

12.156.1 Detailed Description

The \texttt{Fl\_Valuator} class controls a single floating-point value and provides a consistent interface to set the value, range, and step, and insures that callbacks are done the same for every object.

There are probably more of these classes in FLTK than any others:
In the above diagram each box surrounds an actual subclass. These are further differentiated by setting the `type()` of the widget to the symbolic value labeling the widget. The ones labelled "0" are the default versions with a type(0). For consistency the symbol FL_VERTICAL is defined as zero.

### 12.156.2 Constructor & Destructor Documentation

#### 12.156.2.1 `Fl_Valuator()`

```cpp
Fl_Valuator::Fl_Valuator ( int X, int Y, int W, int H, const char * L ) [protected]
```

Creates a new `Fl_Valuator` widget using the given position, size, and label string. The default boxtype is FL_NO_BOX.

### 12.156.3 Member Function Documentation

#### 12.156.3.1 `format()`

```cpp
int Fl_Valuator::format ( char * buffer ) [virtual]
```

Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter. The actual format used depends on the current step value. If the step value has been set to zero then a %g format is used. If the step value is non-zero, then a %f format is used, where the precision is calculated to show sufficient digits for the current step value. An integer step value, such as 1 or 1.0, gives a precision of 0, so the formatted value will appear as an integer.

This method is used by the `Fl_Valuator_...` group of widgets to format the current value into a text string. The return value is the length of the formatted text. The formatted value is written into `buffer`. `buffer` should have space for at least 128 bytes.

You may override this function to create your own text formatting.

#### 12.156.3.2 `increment()`

```cpp
double Fl_Valuator::increment ( )
```
double v,
int n )
Adds n times the step value to the passed value.
If step was set to zero it uses fabs(maximum() - minimum()) / 100.

12.156.3.3 maximum() [1/2]

double Fl_Valuator::maximum ( ) const [inline]
Gets the maximum value for the valuator.

12.156.3.4 maximum() [2/2]

void Fl_Valuator::maximum ( 
    double a ) [inline]
Sets the maximum value for the valuator.

12.156.3.5 minimum() [1/2]

double Fl_Valuator::minimum ( ) const [inline]
Gets the minimum value for the valuator.

12.156.3.6 minimum() [2/2]

void Fl_Valuator::minimum ( 
    double a ) [inline]
Sets the minimum value for the valuator.

12.156.3.7 precision()

void Fl_Valuator::precision ( 
    int digits )
Sets the step value to 1.0 / 10^digits.
Precision digits is limited to 0...9 to avoid internal overflow errors. Values outside this range are clamped.

Note
For negative values of digits the step value is set to A = 1.0 and B = 1, i.e. 1.0/1 = 1.

12.156.3.8 range()

void Fl_Valuator::range ( 
    double a,
    double b ) [inline]
Sets the minimum and maximum values for the valuator.
When the user manipulates the widget, the value is limited to this range. This clamping is done after rounding to the step value (this makes a difference if the range is not a multiple of the step).
The minimum may be greater than the maximum. This has the effect of "reversing" the object so the larger values are in the opposite direction. This also switches which end of the filled sliders is filled.
Some widgets consider this a "soft" range. This means they will stop at the range, but if the user releases and grabs the control again and tries to move it further, it is allowed.
The range may affect the display. You must redraw() the widget after changing the range.
12.156.3.9  round()

double Fl_Valuator::round (double v)
Round the passed value to the nearest step increment.
Does nothing if step is zero.

12.156.3.10  step()

double Fl_Valuator::step ( ) const [inline]
Gets or sets the step value.
As the user moves the mouse the value is rounded to the nearest multiple of the step value. This is done before clamping it to the range. For most widgets the default step is zero.
For precision the step is stored as the ratio of a double A and an integer B = A/B. You can set these values directly. Currently setting a floating point value sets the nearest A/1 or 1/B value possible.

12.156.3.11  value() [1/2]

double Fl_Valuator::value ( ) const [inline]
Gets the floating point(double) value.
See int value(double)

12.156.3.12  value() [2/2]

int Fl_Valuator::value (double v)
Sets the current value.
The new value is not clamped or otherwise changed before storing it. Use clamp() or round() to modify the value before calling value(). The widget is redrawn if the new value is different than the current one. The initial value is zero.
changed() will return true if the user has moved the slider, but it will be turned off by value(x) and just before doing a callback (the callback can turn it back on if desired).

12.156.3.13  value_damage()

void Fl_Valuator::value_damage ( ) [protected], [virtual]
Asks for partial redraw.
Reimplemented in Fl_Adjuster.
The documentation for this class was generated from the following files:

- Fl_Valuator.H
- Fl_Valuator.cxx

12.157  Fl_Value_Input Class Reference

The Fl_Value_Input widget displays a numeric value.
#include <Fl_Value_Input.H>
Inheritance diagram for Fl_Value_Input:
Public Member Functions

- **FL_Color cursor_color () const**
  
  Gets the color of the text cursor.

- **void cursor_color (FL_Color n)**
  
  Sets the color of the text cursor.

- **FL_Value_Input (int x, int y, int w, int h, const char ∗l=0)**
  
  Creates a new FL_Value_Input widget using the given position, size, and label string.

- **int handle (int) FL_OVERRIDE**

  Handles the specified event.

- **void resize (int, int, int, int) FL_OVERRIDE**

  Changes the size or position of the widget.

- **int shortcut () const**

  Returns the current shortcut key for the Input.

- **void shortcut (int s)**

  Sets the shortcut key to s.

- **char soft () const**

  If "soft" is turned on, the user is allowed to drag the value outside the range.

- **void soft (char s)**

  See void FL_Value_Input::soft(char s)

- **FL_Color textcolor () const**

  Gets the color of the text in the value box.

- **void textcolor (FL_Color n)**

  Sets the color of the text in the value box.

- **FL_Font textfont () const**

  Gets the typeface of the text in the value box.

- **void textfont (FL_Font s)**

  Sets the typeface of the text in the value box.

- **FL_Fontsize textsize () const**

  Gets the size of the text in the value box.

- **void textsize (FL_Fontsize s)**

  Sets the size of the text in the value box.

Public Member Functions inherited from FL_Valuator

- **void bounds (double a, double b)**

  Sets the minimum (a) and maximum (b) values for the valuator widget.

- **double clamp (double)**

  Clamps the passed value to the valuator range.

- **virtual int format (char ∗)**

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.

- **double increment (double, int)**

  Adds n times the step value to the passed value.

- **double maximum () const**

  Gets the maximum value for the valuator.

- **void maximum (double a)**

  Sets the maximum value for the valuator.

- **double minimum () const**

  Gets the minimum value for the valuator.

- **void minimum (double a)**

  Sets the minimum value for the valuator.

- **void precision (int digits)**
Sets the step value to $1.0 / 10^{10}$.

- void **range** (double a, double b)
  
  Sets the minimum and maximum values for the valuator.

- double **round** (double)
  
  Round the passed value to the nearest step increment.

- double **step** () const
  
  Gets or sets the step value.

- void **step** (double a, int b)
  
  See double **Fl_Valuator::step()** const

- void **step** (double s)
  
  See double **Fl_Valuator::step()** const.

- void **step** (int a)
  
  See double **Fl_Valuator::step()** const

- double **value** () const
  
  Gets the floating point(double) value.

- int **value** (double)
  
  Sets the current value.

**Public Member Functions inherited from Fl_Widget**

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void **activate** ()
  
  Activates the widget.
- unsigned int **active** () const
  
  Returns whether the widget is active.
- int **active_r** () const
  
  Returns whether the widget and all of its parents are active.
- **Fl_Align** **align** () const
  
  Gets the label alignment.
- void **align** (**Fl_Align** alignment)
  
  Sets the label alignment.
- long **argument** () const
  
  Gets the current user data (long) argument that is passed to the callback function.
- void **argument** (long v)
  
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class **Fl_GL_Window** ∗ **as_gl_window** ()
  
  Returns an **Fl_GL_Window** pointer if this widget is an **Fl_GL_Window**.
- virtual class **Fl_GL_Window** const ∗ **as_gl_window** ()

- virtual class **Fl_Group** ∗ **as_group** ()
  
  Returns an **Fl_Group** pointer if this widget is an **Fl_Group**.
- virtual **Fl_Group** const ∗ **as_group** ()
- virtual **Fl_Window** ∗ **as_window** ()
  
  Returns an **Fl_Window** pointer if this widget is an **Fl_Window**.
- virtual **Fl_Window** const ∗ **as_window** ()

- void **bind_deimage** (**Fl_Image** ∗ img)
  
  Sets the image to use as part of the widget label when in the inactive state.
- void **bind_deimage** (int f)
  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void **bind_image** (**Fl_Image** ∗ img)
Sets the image to use as part of the widget label when in the active state.

- void **bind_image** (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- **Fl_Boxtype** box () const
  Gets the box type of the widget.

- void box (**Fl_Boxtype** new_box)
  Sets the box type for the widget.

- **Fl_Callback_p** callback () const
  Gets the current callback function for the widget.

- void callback (**Fl_Callback** +cb)
  Sets the current callback function for the widget.

- void callback (**Fl_Callback** +cb, **Fl_Callback_User_Data** +p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

- void callback (**Fl_Callback** +cb, void +p)
  Sets the current callback function and data for the widget.

- void callback (**Fl_Callback0** +cb)
  Sets the current callback function for the widget.

- void callback (**Fl_Callback1** +cb, long p=0)
  Sets the current callback function for the widget.

- unsigned int changed () const
  Checks if the widget value changed since the last callback.

- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  Clears or sets the damage flags.

- void clear_output ()
  Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.

- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

- **Fl_Color** color () const
  Gets the background color of the widget.

- void color (**Fl_Color** bg)
  Sets the background color of the widget.

- void color (**Fl_Color** bg, **Fl_Color** sel)
  Sets the background and selection color of the widget.

- **Fl_Color** color2 () const
  For back compatibility only.

- void color2 (unsigned a)
  For back compatibility only.

- int contains (**const** **Fl_Widget** +w) const
  Checks if w is a child of this widget.

- void copy_label (**const** char +new_label)
  Sets the current label.

- void copy_tooltip (**const** char +text)
  Sets the current tooltip text.

- uchar damage () const
  Returns non-zero if **draw()** needs to be called.
- void damage (uchar c)
  
  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)
  
  Sets the damage bits for an area inside the widget.
- int damage_resize (int, int, int)
  
  Internal use only.
- void deactivate ()
  
  Deactivates the widget.
- Fl_Image * deimage ()
  
  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const
  
  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const
  
  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  
  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const
  
  Gets the widget height.
- virtual void hide ()
  
  Makes a widget invisible.
- Fl_Image * image ()
  
  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const
  
  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const
  
  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const
  
  Checks if this widget is a child of wgt.
- int is_label_copied () const
  
  Returns whether the current label was assigned with copy_label().
- const char * label () const
  
  Gets the current label text.
- void label (const char *text)
  
  Sets the current label text.
- void label (Fl_Labeltype a, const char *b)
  
  Shortcut to set the label text and type in one call.
- Fl_Color labelcolor () const
Gets the label color.

- `void labelcolor (Fl_Color c)`
  Sets the label color.

- `Fl_Font labelfont () const`
  Gets the font to use.

- `void labelfont (Fl_Font f)`
  Sets the font to use.

- `Fl_Fontsize labelsize () const`
  Gets the font size in pixels.

- `void labelsize (Fl_Fontsize pix)`
  Sets the font size in pixels.

- `Fl_Labeltype labeltype () const`
  Gets the label type.

- `void labeltype (Fl_Labeltype a)`
  Sets the label type.

- `void measure_label (int &ww, int &hh) const`
  Sets width `ww` and height `hh` accordingly with the label size.

- `bool needs_keyboard () const`
  Returns whether this widget needs a keyboard.

- `void needs_keyboard (bool needs)`
  Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  Returns if a widget is used for output only.

- `Fl_Group * parent () const`
  Returns a pointer to the parent widget.

- `void parent (Fl_Group * p)`
  Internal use only - "for hacks only".

- `void position (int X, int Y)`
  Repositions the window or widget.

- `void redraw ()`
  Schedules the drawing of the widget.

- `void redraw_label ()`
  Schedules the drawing of the label.

- `Fl_Color selection_color () const`
  Gets the selection color.

- `void selection_color (Fl_Color a)`
  Sets the selection color.

- `void set_active ()`
  Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  Marks the value of the widget as changed.

- `void set_output ()`
  Sets a widget to output only.

- `void set_visible ()`
  Makes the widget visible.

- `void set_visible_focus ()`
  Enables keyboard focus navigation with this widget.

- `int shortcut_label () const`
  Returns whether the widget's label uses '\&' to indicate shortcuts.

- `void shortcut_label (int value)`
  Sets whether the widget's label uses '\&' to indicate shortcuts.
• virtual void show ()
    Makes a widget visible.
• void size (int W, int H)
    Changes the size of the widget.
• int take_focus ()
    Gives the widget the keyboard focus.
• unsigned int takesevents () const
    Returns if the widget is able to take events.
• int test_shortcut ()
    Returns true if the widget's label contains the entered 'x' shortcut.
• const char * tooltip () const
    Gets the current tooltip text.
• void tooltip (const char *text)
    Sets the current tooltip text.
• Fl_Window * top_window () const
    Returns a pointer to the top-level window for the widget.
• Fl_Window * top_window_offset (int &xoff, int &yoff) const
    Finds the x/y offset of the current widget relative to the top-level window.
• uchar type () const
    Gets the widget type.
• void type (uchar t)
    Sets the widget type.
• int use_accents_menu ()
    Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
• void * user_data () const
    Gets the user data for this widget.
• void user_data (Fl_Callback_User_Data *v, bool auto_free)
    Sets the user data for this widget.
• void user_data (void *v)
    Sets the user data for this widget.
• unsigned int visible () const
    Returns whether a widget is visible.
• unsigned int visible_focus () const
    Checks whether this widget has a visible focus.
• void visible_focus (int v)
    Modifies keyboard focus navigation.
• int visible_r () const
    Returns whether a widget and all its parents are visible.
• int w () const
    Gets the widget width.
• Fl_When when () const
    Returns the conditions under which the callback is called.
• void when (uchar i)
    Sets the flags used to decide when a callback is called.
• Fl_Window * window () const
    Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
    Gets the widget position in its window.
• int y () const
    Gets the widget position in its window.
• virtual ~Fl_Widget ()
    Destroys the widget.
Public Attributes

- **Fl_Input** input

Protected Member Functions

- void **draw ()** FL_OVERRIDE
  
  Draws the widget.

Protected Member Functions inherited from **Fl_Valuator**

- **Fl_Valuator** (int X, int Y, int W, int H, const char ∗L)
  
  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)
  
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push ()**
  
  Stores the current value in the previous value.
- void **handle_release ()**
  
  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal ()** const
  
  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value ()** const
  
  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)
  
  Sets the current floating point value.
- double **softclamp** (double)
  
  Clamps the value, but accepts v if the previous value is not already out of range.

Protected Member Functions inherited from **Fl_Widget**

- void **clear_flag** (unsigned int c)
  
  Clears a flag in the flags mask.
- void **draw_backdrop ()** const
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box ()** const
  
  Draws the widget box according its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const
  
  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void **draw_focus ()** const
  
  Draws a focus rectangle around the widget.
- void **draw_focus** (Fl_Boxtype t, int X, int Y, int W, int H) const
  
  Draws a focus rectangle around the widget.
- void **draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.
- void **draw_label ()** const
  
  Draws the widget's label at the defined label position.
- void **draw_label** (int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.
- **Fl_Widget** (int x, int y, int w, int h, const char ∗label=0L)
  
  Creates a widget at the given position and size.
- unsigned int **flags ()** const
12.157 Fl_Value_Input Class Reference

--

**Gets the widget flags mask.**

- **void h (int v)**
  
  *Internal use only.*

- **void set_flag (unsigned int c)**
  
  *Sets a flag in the flags mask.*

- **void w (int v)**
  
  *Internal use only.*

- **void x (int v)**
  
  *Internal use only.*

- **void y (int v)**
  
  *Internal use only.*

**Additional Inherited Members**

**Static Public Member Functions inherited from Fl_Widget**

- **static void default_callback (Fl_Widget *widget, void *data)**
  
  *The default callback for all widgets that don't set a callback.*

- **static unsigned int label_shortcut (const char *t)**
  
  *Returns the Unicode value of the '&x' shortcut in a given text.*

- **static int test_shortcut (const char *, const bool require_alt=false)**
  
  *Returns true if the given text t contains the entered 'x' shortcut.*

**Protected Types inherited from Fl_Widget**

- **enum {**

  | INACTIVE | 1<<0 |
  | INVISIBLE | 1<<1 |
  | OUTPUT | 1<<2 |
  | NOBORDER | 1<<3 |
  | FORCE_POSITION | 1<<4 |
  | NON_MODAL | 1<<5 |
  | SHORTCUT_LABEL | 1<<6 |
  | CHANGED | 1<<7 |
  | OVERRIDE | 1<<8 |
  | VISIBLE_FOCUS | 1<<9 |
  | COPIED_LABEL | 1<<10 |
  | CLIP_CHILDREN | 1<<11 |
  | MENU_WINDOW | 1<<12 |
  | TOOLTIP_WINDOW | 1<<13 |
  | MODAL | 1<<14 |
  | NO_OVERLAY | 1<<15 |
  | GROUP_RELATIVE | 1<<16 |
  | COPIED_TOOLTIP | 1<<17 |
  | FULLSCREEN | 1<<18 |
  | MAC_USE_ACCENTS_MENU | 1<<19 |
  | NEEDS_KEYBOARD | 1<<20 |
  | IMAGE_BOUND | 1<<21 |
  | DEIMAGE_BOUND | 1<<22 |
  | AUTO_DELETE_USER_DATA | 1<<23 |
  | MAXIMIZED | 1<<24 |
  | USERFLAG3 | 1<<29 |
  | USERFLAG2 | 1<<30 |
  | USERFLAG1 | 1<<31 |

  *

  *flags possible values enumeration.*

---

**12.157.1 Detailed Description**

The **Fl_Value_Input** widget displays a numeric value. The user can click in the text field and edit it - there is in fact a hidden **Fl_Input** widget with type(FL_FLOAT_INPUT) or type(FL_INT_INPUT) in there - and when they hit return or tab the value updates to what they typed and the callback is done.

If **step()** is non-zero and integral, then the range of numbers is limited to integers instead of floating point numbers. As well as displaying the value as an integer, typed input is also limited to integer values, even if the hidden **Fl_Input** widget is of type(FL_FLOAT_INPUT).

If **step()** is non-zero, the user can also drag the mouse across the object and thus slide the value. The left button moves one **step()** per pixel, the middle by 10 **step()**, and the right button by 100 * step(). It is therefore impossible to select text by dragging across it, although clicking can still move the insertion cursor.

If **step()** is non-zero and integral, then the range of numbers are limited to integers instead of floating point values.
12.157.2 Constructor & Destructor Documentation

12.157.2.1 Fl_Value_Input()

Fl_Value_Input::Fl_Value_Input (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char ∗ l = 0 )

Creates a new Fl_Value_Input widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX.

12.157.3 Member Function Documentation

12.157.3.1 cursor_color() [1/2]

Fl_Color Fl_Value_Input::cursor_color ( ) const [inline]

Gets the color of the text cursor.
The text cursor is black by default.

12.157.3.2 cursor_color() [2/2]

void Fl_Value_Input::cursor_color (  
  Fl_Color n ) [inline]

Sets the color of the text cursor.
The text cursor is black by default.

12.157.3.3 draw()

void Fl_Value_Input::draw ( ) [protected], [virtual]

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:
Fl_Widget ∗s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar  
s->draw();  // calls Fl_Scrollbar::draw()

Implements Fl_Widget.

12.157.3.4 handle()

int Fl_Value_Input::handle (  
  int event ) [virtual]

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class. In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in | event | the kind of event received |

Return values

| 0 | if the event was not used or understood |
| 1 | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Widget.

12.157.3.5 resize()

```cpp
void Fl_Value_Input::resize (  
    int x,  
    int y,  
    int w,  
    int h ) [virtual]
```

Changes the size or position of the widget.

This is a virtual function so that the widget may implement its own handling of resizing. The default version does not call the redraw() method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position.

Some window managers under X11 call resize() a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations.

position(X, Y) is a shortcut for resize(X, Y, w(), h()), and size(W, H) is a shortcut for resize(x(), y(), W, H).

Parameters

| in | x,y | new position relative to the parent window |
| in | w,h | new size |

See also

position(int,int), size(int,int)

Reimplemented from Fl_Widget.

12.157.3.6 shortcut() [1/2]

```cpp
int Fl_Value_Input::shortcut ( ) const [inline]
```

Returns the current shortcut key for the Input.

See also

Fl_Value_Input::shortcut(int)
12.157.3.7 shortcut() [2/2]

```cpp
void Fl_Value_Input::shortcut ( int s ) [inline]
```

Sets the shortcut key to `s`.
Setting this overrides the use of '&' in the `label()`. The value is a bitwise OR of a key and a set of shift flags, for example `FL_ALT | 'a'`, `FL_ALT | (FL_F + 10)`, or just 'a'. A value of 0 disables the shortcut.
The key can be any value returned by `Fl::event_key()`, but will usually be an ASCII letter. Use a lower-case letter unless you require the shift key to be held down.
The shift flags can be any set of values accepted by `Fl::event_state()`. If the bit is on that shift key must be pushed. Meta, Alt, Ctrl, and Shift must be off if they are not in the shift flags (zero for the other bits indicates a "don't care" setting).

12.157.3.8 soft()

```cpp
char Fl_Value_Input::soft ( ) const [inline]
```

If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
The default is true.

12.157.3.9 textcolor()

```cpp
Fl_Color Fl_Value_Input::textcolor ( ) const [inline]
```

Gets the color of the text in the value box.

12.157.3.10 textfont() [1/2]

```cpp
Fl_Font Fl_Value_Input::textfont ( ) const [inline]
```

Gets the typeface of the text in the value box.

12.157.3.11 textfont() [2/2]

```cpp
void Fl_Value_Input::textfont ( Fl_Font s ) [inline]
```

Sets the typeface of the text in the value box.

12.157.3.12 textsize() [1/2]

```cpp
Fl_Fontsize Fl_Value_Input::textsize ( ) const [inline]
```

Gets the size of the text in the value box.

12.157.3.13 textsize() [2/2]

```cpp
void Fl_Value_Input::textsize ( Fl_Fontsize s ) [inline]
```

Sets the size of the text in the value box.

The documentation for this class was generated from the following files:

- Fl_Value_Input.H
- Fl_Value_Input.cxx
### 12.158  Fl_Value_Output Class Reference

The **Fl_Value_Output** widget displays a floating point value. 

```c
#include <Fl_Value_Output.H>
```

Inheritance diagram for Fl_Value_Output:

```
Fl_Widget
   ^
   |  
Fl_Valuator
     ^
     |  
Fl_Value_Output
```

#### Public Member Functions

- **Fl_Value_Output** (int x, int y, int w, int h, const char ∗l=0)
  
  Creates a new **Fl_Value_Output** widget using the given position, size, and label string.

- **int handle** (int) **FL_OVERRIDE**
  
  Handles the specified event.

- **uchar soft** () const
  
  If "soft" is turned on, the user is allowed to drag the value outside the range.

- **void soft** (uchar s)
  
  If "soft" is turned on, the user is allowed to drag the value outside the range.

- **Fl_Color textcolor** () const
  
  Sets the color of the text in the value box.

- **void textcolor** (Fl_Color s)
  
  Gets the color of the text in the value box.

- **Fl_Font textfont** () const
  
  Gets the typeface of the text in the value box.

- **void textfont** (Fl_Font s)
  
  Sets the typeface of the text in the value box.

- **Fl_Fontsize textsize** () const
  
  Gets the size of the text in the value box.

- **void textsize** (Fl_Fontsize s)

#### Public Member Functions inherited from **Fl_Valuator**

- **void bounds** (double a, double b)
  
  Sets the minimum (a) and maximum (b) values for the valuator widget.

- **double clamp** (double)
  
  Clamps the passed value to the valuator range.

- **virtual int format** (char ∗)
  
  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.

- **double increment** (double, int)
  
  Adds n times the step value to the passed value.

- **double maximum** () const
  
  Gets the maximum value for the valuator.

- **void maximum** (double a)
  
  Sets the maximum value for the valuator.

- **double minimum** () const
  
  Gets the minimum value for the valuator.
• void **minimum**(double a)
  
  Sets the minimum value for the valuator.

• void **precision**(int digits)
  
  Sets the step value to \(1.0 / 10^{\text{digits}}\).

• void **range**(double a, double b)
  
  Sets the minimum and maximum values for the valuator.

• double **round**(double)
  
  Round the passed value to the nearest step increment.

• double **step**(const)
  
  Gets or sets the step value.

• void **step**(double a, int b)
  
  See double Fl_Valuator::step() const

• void **step**(double s)
  
  See double Fl_Valuator::step() const.

• void **step**(int a)
  
  See double Fl_Valuator::step() const

• double **value**(const)
  
  Gets the floating point(double) value.

• int **value**(double)
  
  Sets the current value.

**Public Member Functions inherited from Fl_Widget**

• void **_clear_fullscreen**(const)
  
  Activates the widget.

• unsigned int **active**(const)
  
  Returns whether the widget is active.

• int **active_r**(const)
  
  Returns whether the widget and all of its parents are active.

• Fl_Align **align**(const)
  
  Gets the label alignment.

• void **align**(Fl_Align alignment)
  
  Sets the label alignment.

• long **argument**(const)
  
  Gets the current user data (long) argument that is passed to the callback function.

• void **argument**(long v)
  
  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window **as_gl_window**(const)
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window **as_gl_window**(const)
  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual Fl_Group **as_group**(const)
  
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Group **as_group**(const)
  
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Window **as_window**(const)
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window **as_window**(const)
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual bind_deimage (Fl_Image *img)
  
  Sets the image to use as part of the widget label when in the inactive state.
**void bind_deimage (int f)**

Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

**void bind_image (Fl_Image *img)**

Sets the image to use as part of the widget label when in the active state.

**void bind_image (int f)**

Bind the image to the widget, so the widget will delete the image when it is no longer needed.

**Fl_Boxtype box () const**

Gets the box type of the widget.

**void box (Fl_Boxtype new_box)**

Sets the box type for the widget.

**Fl_Callback_p callback () const**

Gets the current callback function for the widget.

**void callback (Fl_Callback *cb)**

Sets the current callback function for the widget.

**void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)**

Sets the current callback function and managed user data for the widget.

**void callback (Fl_Callback *cb, void *p)**

Sets the current callback function and data for the widget.

**void callback (Fl_Callback0 *cb)**

Sets the current callback function for the widget.

**void callback (Fl_Callback1 *cb, long p=0)**

Sets the current callback function for the widget.

**unsigned int changed () const**

Checks if the widget value changed since the last callback.

**void clear_active ()**

Marks the widget as inactive without sending events or changing focus.

**void clear_changed ()**

Marks the value of the widget as unchanged.

**void clear_damage (uchar c=0)**

Clears or sets the damage flags.

**void clear_output ()**

Sets a widget to accept input.

**void clear_visible ()**

Hides the widget.

**void clear_visible_focus ()**

Disables keyboard focus navigation with this widget.

**Fl_Color color () const**

Gets the background color of the widget.

**void color (Fl_Color bg)**

Sets the background color of the widget.

**void color (Fl_Color bg, Fl_Color sel)**

Sets the background and selection color of the widget.

**Fl_Color color2 () const**

For back compatibility only.

**void color2 (unsigned a)**

For back compatibility only.

**int contains (const Fl_Widget *w) const**

Checks if w is a child of this widget.

**void copy_label (const char *new_label)**

Sets the current label.

**void copy_tooltip (const char *text)**

Generated by Doxygen
Sets the current tooltip text.

- **uchar damage () const**
  
  Returns non-zero if `draw()` needs to be called.

- **void damage (uchar c)**
  
  Sets the damage bits for the widget.

- **void damage (uchar c, int x, int y, int w, int h)**
  
  Sets the damage bits for an area inside the widget.

- **int damage_resize (int, int, int, int)**
  
  Internal use only.

- **void deactivate ()**
  
  Deactivates the widget.

- **Fl_Image * deimage ()**
  
  Gets the image that is used as part of the widget label when in the inactive state.

- **const Fl_Image * deimage () const**
  
  Gets the image that is used as part of the widget label when in the inactive state.

- **void deimage (Fl_Image &img)**
  
  Sets the image to use as part of the widget label when in the inactive state.

- **void deimage (Fl_Image *img)**
  
  Sets the image to use as part of the widget label when in the inactive state.

- **int deimage_bound () const**
  
  Returns whether the inactive image is managed by the widget.

- **void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  
  Calls the widget callback function with default arguments.

- **void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  
  Calls the widget callback function with arbitrary arguments.

- **void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)**
  
  Calls the widget callback function with arbitrary arguments.

- **void draw_label (int, int, int, int, Fl_Align) const**
  
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h () const**
  
  Gets the widget height.

- **virtual void hide ()**
  
  Makes a widget invisible.

- **Fl_Image * image ()**
  
  Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image * image () const**
  
  Gets the image that is used as part of the widget label when in the active state.

- **void image (Fl_Image &img)**
  
  Sets the image to use as part of the widget label when in the active state.

- **void image (Fl_Image *img)**
  
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const**
  
  Returns whether the image is managed by the widget.

- **int inside (const Fl_Widget *wgt) const**
  
  Checks if this widget is a child of `wgt`.

- **int is_label_copied () const**
  
  Returns whether the current label was assigned with `copy_label()`.

- **const char * label () const**
  
  Gets the current label text.

- **void label (const char *text)**
  
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  
  Shortcut to set the label text and type in one call.

• Fl_Color labelcolor () const
  
  Gets the label color.

• void labelcolor (Fl_Color c)
  
  Sets the label color.

• Fl_Font labelfont () const
  
  Gets the font to use.

• void labelfont (Fl_Font f)
  
  Sets the font to use.

• Fl_Fontsize labelsize () const
  
  Gets the font size in pixels.

• void labelsize (Fl_Fontsize pix)
  
  Sets the font size in pixels.

• Fl_Labeltype labeltype () const
  
  Gets the label type.

• void labeltype (Fl_Labeltype a)
  
  Sets the label type.

• void measure_label (int &ww, int &hh) const
  
  Sets width ww and height hh accordingly with the label size.

• bool needs_keyboard () const
  
  Returns whether this widget needs a keyboard.

• void needs_keyboard (bool needs)
  
  Sets whether this widget needs a keyboard.

• unsigned int output () const
  
  Returns if a widget is used for output only.

• Fl_Group * parent () const
  
  Returns a pointer to the parent widget.

• void parent (Fl_Group *p)
  
  Internal use only - "for hacks only".

• void position (int X, int Y)
  
  Repositions the window or widget.

• void redraw ()
  
  Schedules the drawing of the widget.

• void redraw_label ()
  
  Schedules the drawing of the label.

• virtual void resize (int x, int y, int w, int h)
  
  Changes the size or position of the widget.

• Fl_Color selection_color () const
  
  Gets the selection color.

• void selection_color (Fl_Color a)
  
  Sets the selection color.

• void set_active ()
  
  Marks the widget as active without sending events or changing focus.

• void set_changed ()
  
  Marks the value of the widget as changed.

• void set_output ()
  
  Sets a widget to output only.

• void set_visible ()
  
  Makes the widget visible.

• void set_visible_focus ()
  
  

Generated by Doxygen
Enables keyboard focus navigation with this widget.

- int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.

- void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.

- virtual void show ()
  Makes a widget visible.

- void size (int W, int H)
  Changes the size of the widget.

- int take_focus ()
  Gives the widget the keyboard focus.

- unsigned int takesevents () const
  Returns if the widget is able to take events.

- int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.

- const char * tooltip () const
  Gets the current tooltip text.

- void tooltip (const char * text)
  Sets the current tooltip text.

- Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.

- Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const
  Gets the widget type.

- void type (uchar t)
  Sets the widget type.

- int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void * user_data () const
  Gets the user data for this widget.

- void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

- void user_data (void *v)
  Sets the user data for this widget.

- unsigned int visible () const
  Returns whether a widget is visible.

- unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

- void visible_focus (int v)
  Modifies keyboard focus navigation.

- int visible_r () const
  Returns whether a widget and all its parents are visible.

- int w () const
  Gets the widget width.

- Fl_When when () const
  Returns the conditions under which the callback is called.

- void when (uchar i)
  Sets the flags used to decide when a callback is called.

- Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  
  Gets the widget position in its window.

• int y () const
  
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  
  Destroys the widget.

Protected Member Functions

• void draw () FL_OVERRIDE

  Draws the widget.

Protected Member Functions inherited from Fl_Valuator

• Fl_Valuator (int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.

• void handle_drag (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.

• void handle_push ()

  Stores the current value in the previous value.

• void handle_release ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.

• int horizontal () const

  Tells if the valuator is an FL_HORIZONTAL one.

• double previous_value () const

  Gets the previous floating point value before an event changed it.

• void set_value (double v)

  Sets the current floating point value.

• double softclamp (double)

  Clamps the value, but accepts v if the previous value is not already out of range.

• virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

• void clear_flag (unsigned int c)

  Clears a flag in the flags mask.

• void draw_backdrop () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• void draw_box () const

  Draws the widget box according its box style.

• void draw_box (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.

• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.

• void draw_focus () const

  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.

• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.

• void draw_label () const

  Draws the label of the widget.
Draws the widget's label at the defined label position.

- void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)
  Creates a widget at the given position and size.
- unsigned int flags () const
  Gets the widget flags mask.
- void h (int v)
  Internal use only.
- void set_flag (unsigned int c)
  Sets a flag in the flags mask.
- void w (int v)
  Internal use only.
- void x (int v)
  Internal use only.
- void y (int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

12.158.1 Detailed Description

The Fl_Value_Output widget displays a floating point value. If step() is not zero, the user can adjust the value by dragging the mouse left and right. The left button moves one step() per pixel, the middle by 10 * step(), and the right button by 100 * step(). This is much lighter-weight than Fl_Value_Input because it contains no text editing code or character buffer.
12.158.2 Constructor & Destructor Documentation

12.158.2.1 Fl_Value_Output()

Fl_Value_Output::Fl_Value_Output (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char ∗ l = 0 )  

Creates a new Fl_Value_Output widget using the given position, size, and label string.  
The default boxtype is FL_NO_BOX.  
Inherited destructor destroys the Valuator.

12.158.3 Member Function Documentation

12.158.3.1 draw()

void Fl_Value_Output::draw ( ) [protected], [virtual]  

Draws the widget.  
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.  
Override this function to draw your own widgets.  
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:  
Fl_Widget ∗s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar  
s->draw(); // calls Fl_Scrollbar::draw()  
Implements Fl_Widget.

12.158.3.2 handle()

int Fl_Value_Output::handle (  
    int event ) [virtual]  

Handles the specified event.  
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.  
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.  
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.  
One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class.  
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

Parameters

| in | event | the kind of event received |

Return values

| 0 | if the event was not used or understood |
Return values

\[ \text{if the event was used and can be deleted} \]

See also

Fl_Event

Reimplemented from Fl_Widget.

12.158.3.3  soft() [1/2]

uchar Fl_Value_Output::soft() const [inline]
If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

12.158.3.4  soft() [2/2]

void Fl_Value_Output::soft(uchar s) [inline]
If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

12.158.3.5  textcolor() [1/2]

Fl_Color Fl_Value_Output::textcolor() const [inline]
Sets the color of the text in the value box.

12.158.3.6  textcolor() [2/2]

void Fl_Value_Output::textcolor(Fl_Color s) [inline]
Gets the color of the text in the value box.

12.158.3.7  textfont() [1/2]

Fl_Font Fl_Value_Output::textfont() const [inline]
Gets the typeface of the text in the value box.

12.158.3.8  textfont() [2/2]

void Fl_Value_Output::textfont(Fl_Font s) [inline]
Sets the typeface of the text in the value box.

12.158.3.9  textsize()

Fl_Fontsize Fl_Value_Output::textsize() const [inline]
Gets the size of the text in the value box.
The documentation for this class was generated from the following files:

- Fl_Value_Output.H
- Fl_Value_Output.cxx

12.159 Fl_Value_Slider Class Reference

The Fl_Value_Slider widget is a Fl_Slider widget with a box displaying the current value.

#include <Fl_Value_Slider.H>

Inheritance diagram for Fl_Value_Slider:

```
Fl_Widget
   |
   |- Fl_Valuator
      |
      |       Fl_Slider
      |         |
      |         |       Fl_Value_Slider
      |         |           |
      |         |           |   Fl_Hor_Value_Slider
```

Public Member Functions

- **Fl_Value_Slider** (int x, int y, int w, int h, const char ∗l=0)
  
  * Creates a new Fl_Value_Slider widget using the given position, size, and label string.

- int **handle** (int) FL_OVERRIDE
  
  * Handles the specified event.

- **Fl_Color** textcolor () const
  
  * Gets the color of the text in the value box.

- void **textcolor** (Fl_Color s)
  
  * Sets the color of the text in the value box.

- **Fl_Font** textfont () const
  
  * Gets the typeface of the text in the value box.

- void **textfont** (Fl_Font s)
  
  * Sets the typeface of the text in the value box.

- **Fl_Fontsize** textsize () const
  
  * Gets the size of the text in the value box.

- void **textsize** (Fl_Fontsize s)
  
  * Sets the size of the text in the value box.

- int **value_height** () const
  
  * Gets the height of the value box in pixels (vertical mode only).

- void **value_height** (int s)
  
  * Sets the height of the value box in pixels (vertical mode only).

- int **value_width** () const
  
  * Gets the width of the value box in pixels (horizontal mode only).

- void **value_width** (int s)
  
  * Sets the width of the value box in pixels (horizontal mode only).
Public Member Functions inherited from Fl_Slider

- void bounds (double a, double b)
  Sets the minimum (a) and maximum (b) values for the valuator widget.
- Fl_Slider (int X, int Y, int W, int H, const char *L=0)
  Creates a new Fl_Slider widget using the given position, size, and label string.
- Fl_Slider (uchar t, int X, int Y, int W, int H, const char *L)
  Creates a new Fl_Slider widget using the given type, position, size, and label string.
- int scrollvalue (int pos, int size, int first, int total)
  Sets the size and position of the sliding knob in the box.
- Fl_Boxtype slider () const
  Gets the slider box type.
- void slider (Fl_Boxtype c)
  Sets the slider box type.
- float slider_size () const
  Get the dimensions of the moving piece of slider.
- void slider_size (double v)
  Set the dimensions of the moving piece of slider.

Public Member Functions inherited from Fl_Valuator

- void bounds (double a, double b)
  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double clamp (double)
  Clamps the passed value to the valuator range.
- virtual int format (char *)
  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- double increment (double, int)
  Adds n times the step value to the passed value.
- double maximum () const
  Gets the maximum value for the valuator.
- void maximum (double a)
  Sets the maximum value for the valuator.
- double minimum () const
  Gets the minimum value for the valuator.
- void minimum (double a)
  Sets the minimum value for the valuator.
- void precision (int digits)
  Sets the step value to $1.0 / 10^{\text{digits}}$.
- void range (double a, double b)
  Sets the minimum and maximum values for the valuator.
- double round (double)
  Round the passed value to the nearest step increment.
- double step () const
  Gets or sets the step value.
- void step (double a, int b)
  See double Fl_Valuator::step() const
- void step (double s)
  See double Fl_Valuator::step() const.
- void step (int a)
• double value () const
  Gets the floating point(double) value.

• int value (double)
  Sets the current value.

Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  Activates the widget.

• unsigned int active () const
  Returns whether the widget is active.

• int active_r () const
  Returns whether the widget and all of its parents are active.

• Fl_Align align () const
  Gets the label alignment.

• void align (Fl_Align alignment)
  Sets the label alignment.

• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.

• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const ∗ as_gl_window () const
• virtual Fl_Group ∗ as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Group const ∗ as_group () const
• virtual Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window const ∗ as_window () const
• void bind_deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the inactive state.

• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

• void bind_image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label when in the active state.

• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

• Fl_Boxtype box () const
  Gets the box type of the widget.

• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.

• Fl_Callback_p callback () const
  Gets the current callback function for the widget.

• void callback (Fl_Callback ∗ cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback ∗ cb, Fl_Callback_User_Data ∗ p, bool auto_free)
  Sets the current callback function and managed user data for the widget.
• void callback (Fl_Callback ∗cb, void ∗p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback0 ∗cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 ∗cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
  Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget ∗w) const
  Checks if w is a child of this widget.
• void copy_label (const char ∗new_label)
  Sets the current label.
• void copy_tooltip (const char ∗text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image ∗deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image ∗deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
Sets the image to use as part of the widget label when in the inactive state.

- void deimage(Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound() const
  Returns whether the inactive image is managed by the widget.

- void do_callback(Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

- void do_callback(Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- void do_callback(Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- void draw_label(int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- int h() const
  Gets the widget height.

- virtual void hide()
  Makes a widget invisible.

- Fl_Image *image()
  Gets the image that is used as part of the widget label when in the active state.

- const Fl_Image *image() const
  Gets the image that is used as part of the widget label when in the active state.

- void image(Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.

- void image(Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.

- int image_bound() const
  Returns whether the image is managed by the widget.

- int inside(const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.

- int is_label_copied() const
  Returns whether the current label was assigned with copy_label().

- const char *label() const
  Gets the current label text.

- void label(const char *text)
  Sets the current label pointer.

- void label(Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.

- Fl_Color labelcolor() const
  Gets the label color.

- void labelcolor(Fl_Color c)
  Sets the label color.

- Fl_Font labelfont() const
  Gets the font to use.

- void labelfont(Fl_Font f)
  Sets the font to use.

- Fl_Fontsize labelsize() const
  Gets the font size in pixels.

- void labelsize(Fl_Fontsize pix)
  Sets the font size in pixels.

- Fl_Labeltype labeltype() const
  Gets the label type.
• void labeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - “for hacks only”.
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
• void set_active ()
  Marks the widget as active without sending events or changing focus.
• void set_changed ()
  Marks the value of the widget as changed.
• void set_output ()
  Sets a widget to output only.
• void set_visible ()
  Makes the widget visible.
• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.
• int shortcut_label () const
  Returns whether the widget's label uses '&' to indicate shortcuts.
• void shortcut_label (int value)
  Sets whether the widget's label uses '&' to indicate shortcuts.
• virtual void show ()
  Makes a widget visible.
• void size (int W, int H)
  Changes the size of the widget.
• int take_focus ()
  Gives the widget the keyboard focus.
• unsigned int takesevents () const
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.
• void tooltip (const char ∗text)
  Sets the current tooltip text.

• FL_Window ∗ top_window () const
  Returns a pointer to the top-level window for the widget.

• FL_Window ∗ top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const
  Gets the widget type.

• void type (uchar t)
  Sets the widget type.

• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void ∗ user_data () const
  Gets the user data for this widget.

• void user_data (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.

• void user_data (void ∗v)
  Sets the user data for this widget.

• unsigned int visible () const
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

• void visible_focus (int v)
  Modifies keyboard focus navigation.

• int visible_r () const
  Returns whether a widget and all its parents are visible.

• int w () const
  Gets the widget width.

• FL_When when () const
  Returns the conditions under which the callback is called.

• void when (uchar i)
  Sets the flags used to decide when a callback is called.

• FL_Window ∗ window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  Gets the widget position in its window.

• int y () const
  Gets the widget position in its window.

• virtual ∼Fl_Widget ()
  Destroys the widget.

Protected Member Functions

• void draw () FL_OVERRIDE
  Draws the widget.

Protected Member Functions inherited from Fl_Slider

• void draw (int, int, int, int)

• int handle (int, int, int, int)
Protected Member Functions inherited from Fl_Valuator

- **Fl_Valuator**(int X, int Y, int W, int H, const char \*L)
  
  Creates a new Fl_Valuator widget using the given position, size, and label string.

- **void** handle_drag(double newvalue)
  
  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.

- **void** handle_push()
  
  Stores the current value in the previous value.

- **void** handle_release()
  
  Called after an FL_WHEN_RELEASE event is received and before the callback.

- **int** horizontal() const
  
  Tells if the valuator is an FL_HORIZONTAL one.

- **double** previous_value() const
  
  Gets the previous floating point value before an event changed it.

- **void** set_value(double v)
  
  Sets the current floating point value.

- **double** softclamp(double)
  
  Clamps the value, but accepts v if the previous value is not already out of range.

- **virtual void** value_damage()
  
  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- **void** clear_flag(unsigned int c)
  
  Clears a flag in the flags mask.

- **void** draw_backdrop() const
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- **void** draw_box() const
  
  Draws the widget box according its box style.

- **void** draw_box(Fl_Boxtype t, Fl_Color c) const
  
  Draws a box of type t, of color c at the widget's position and size.

- **void** draw_box(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type t, of color c at the position X,Y and size W,H.

- **void** draw_focus() const
  
  Draws a focus rectangle around the widget.

- **void** draw_focus(Fl_Boxtype t, int X, int Y, int W, int H) const
  
  Draws a focus rectangle around the widget.

- **void** draw_focus(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.

- **void** draw_label() const
  
  Draws the widget's label at the defined label position.

- **void** draw_label(int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.

- **Fl_Widget**(int x, int y, int w, int h, const char \*label=0L)
  
  Creates a widget at the given position and size.

- **unsigned int** flags() const
  
  Gets the widget flags mask.

- **void** h(int v)
  
  Internal use only.

- **void** set_flag(unsigned int c)
  
  Sets a flag in the flags mask.

- **void** w(int v)
Internal use only.

- void x (int v)
  Internal use only.
- void y (int v)
  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22, AUTO_DELETE_USER_DATA = 1<<23,
  MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

12.159.1 Detailed Description

The Fl_Value_Slider widget is a Fl_Slider widget with a box displaying the current value.

![Figure 12.69 Fl_Value_Slider](image)

Figure 12.69 Fl_Value_Slider

12.159.2 Constructor & Destructor Documentation

12.159.2.1 Fl_Value_Slider()

Fl_Value_Slider::Fl_Value_Slider (  
    int X,
    int Y,
    int W,
    int H,
    const char * l = 0 )
1864 Class Documentation

Creates a new Fl_Value_Slider widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX.

12.159.3 Member Function Documentation

12.159.3.1 draw()

```cpp
void Fl_Value_Slider::draw ( ) [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:
```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Slider.

12.159.3.2 handle()

```cpp
int Fl_Value_Slider::handle ( int event ) [virtual]
```

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class. This requires knowledge of the details of the inherited class.

In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.

**Parameters**

<table>
<thead>
<tr>
<th>in event</th>
</tr>
</thead>
<tbody>
<tr>
<td>the kind of event received</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>if the event was not used or understood</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>

See also

Fl_Event

Reimplemented from Fl_Slider.

12.159.3.3 value_height() [1/2]

```cpp
int Fl_Value_Slider::value_height ( ) const [inline]
```

Gets the height of the value box in pixels (vertical mode only).

Since

1.4.0
12.159.3.4 value_height() [2/2]

void Fl_Value_Slider::value_height (
    int s  ) [inline]

Sets the height of the value box in pixels (vertical mode only).
Limited range checking is applied but drawing errors may occur if the size \( s \) is set too high or too low, particularly if
the widget is resized (later).
The programmer is responsible for setting sensible values and widget sizes.
The default value set by the constructor is 25.

Parameters

| in  | s  | new height of the value box |

Since

1.4.0

12.159.3.5 value_width() [1/2]

int Fl_Value_Slider::value_width ( ) const [inline]

Gets the width of the value box in pixels (horizontal mode only).

Since

1.4.0

12.159.3.6 value_width() [2/2]

void Fl_Value_Slider::value_width ( 
    int s  ) [inline]

Sets the width of the value box in pixels (horizontal mode only).
Limited range checking is applied but drawing errors may occur if the size \( s \) is set too high or too low, particularly if
the widget is resized (later).
The programmer is responsible for setting sensible values and widget sizes.
The default value set by the constructor is 35.

Parameters

| in  | s  | new width of the value box |

Since

1.4.0

The documentation for this class was generated from the following files:

- Fl_Value_Slider.H
- Fl_Value_Slider.cxx

12.160 Fl_Widget Class Reference

Fl_Widget is the base class for all widgets in FLTK.

#include <Fl_Widget.H>

Inheritance diagram for Fl_Widget:

Generated by Doxygen
Public Member Functions

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * as_gl_window () const
- virtual Fl_Group * as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * as_group () const
- virtual Fl_Window * as_window ()
  virtual Fl_Window const * as_window () const
Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const & as_window() const
  Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage(Fl_Imagex img)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image(Fl_Imagex img)
  Sets the image to use as part of the widget label when in the active state.

- void bind_image(int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- Fl_Boxtype box() const
  Gets the box type of the widget.

- void box(Fl_Boxtype new_box)
  Sets the box type for the widget.

- Fl_Callback_p callback() const
  Gets the current callback function for the widget.

- void callback(Fl_Callback cb)
  Sets the current callback function for the widget.

- void callback(Fl_Callback cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

- void callback(Fl_Callback cb, void *p)
  Sets the current callback function and data for the widget.

- void callback(Fl_Callback0 cb)
  Sets the current callback function for the widget.

- void callback(Fl_Callback1 cb, long p=0)
  Sets the current callback function for the widget.

- unsigned int changed() const
  Checks if the widget value changed since the last callback.

- void clear_active()
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed()
  Marks the value of the widget as unchanged.

- void clear_damage(uchar c=0)
  Clears or sets the damage flags.

- void clear_output()
  Sets a widget to accept input.

- void clear_visible()
  Hides the widget.

- void clear_visible_focus()
  Disables keyboard focus navigation with this widget.

- Fl_Color color() const
  Gets the background color of the widget.

- void color(Fl_Color bg)
  Sets the background color of the widget.

- void color(Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

- Fl_Color color2() const
  For back compatibility only.

- void color2(unsigned a)
  For back compatibility only.

- int contains(const Fl_Widget *w) const
Checks if w is a child of this widget.

- void copy_label (const char *new_label)
  Sets the current label.

- void copy_tooltip (const char *text)
  Sets the current tooltip text.

- uchar damage () const
  Returns non-zero if draw() needs to be called.

- void damage (uchar c)
  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- int damage_resize (int, int, int, int)
  Internal use only.

- void deactivate ()
  Deactivates the widget.

- Fl_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.

- const Fl_Image *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.

- void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.

- void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const
  Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.

- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.

- virtual void draw ()=0
  Draws the widget.

- void draw_label (int, int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- int h () const
  Gets the widget height.

- virtual int handle (int event)
  Handles the specified event.

- virtual void hide ()
  Makes a widget invisible.

- Fl_Image *image ()
  Gets the image that is used as part of the widget label when in the active state.

- const Fl_Image *image () const
  Gets the image that is used as part of the widget label when in the active state.

- void image (Fl_Image &img)
  Sets the image to use as part of the widget label when in the active state.

- void image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.

- int image_bound () const
  Returns whether the image is managed by the widget.
• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of wgt.
• int is_label_copied () const
  Returns whether the current label was assigned with copy_label().
• const char * label () const
  Gets the current label text.
• void label (const char *text)
  Sets the current label pointer.
• void label (Fl_Labeltype a, const char *b)
  Shortcut to set the label text and type in one call.
• Fl_Color labelcolor () const
  Gets the label color.
• void labelcolor (Fl_Color c)
  Sets the label color.
• Fl_Font labelfont () const
  Gets the font to use.
• void labelfont (Fl_Font f)
  Sets the font to use.
• Fl_Fontsize labelsize () const
  Gets the font size in pixels.
• void labelsize (Fl_Fontsize pix)
  Sets the font size in pixels.
• Fl_Labeltype labellabeltype () const
  Gets the label type.
• void labellabeltype (Fl_Labeltype a)
  Sets the label type.
• void measure_label (int &ww, int &hh) const
  Sets width ww and height hh accordingly with the label size.
• bool needs_keyboard () const
  Returns whether this widget needs a keyboard.
• void needs_keyboard (bool needs)
  Sets whether this widget needs a keyboard.
• unsigned int output () const
  Returns if a widget is used for output only.
• Fl_Group * parent () const
  Returns a pointer to the parent widget.
• void parent (Fl_Group *p)
  Internal use only - “for hacks only”.
• void position (int X, int Y)
  Repositions the window or widget.
• void redraw ()
  Schedules the drawing of the widget.
• void redraw_label ()
  Schedules the drawing of the label.
• virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
• Fl_Color selection_color () const
  Gets the selection color.
• void selection_color (Fl_Color a)
  Sets the selection color.
Marks the widget as active without sending events or changing focus.

• void set_changed ()
  Marks the value of the widget as changed.

• void set_output ()
  Sets a widget to output only.

• void set_visible ()
  Makes the widget visible.

• void set_visible_focus ()
  Enables keyboard focus navigation with this widget.

• int shortcut_label () const
  Returns whether the widget's label uses ' &' to indicate shortcuts.

• void shortcut_label (int value)
  Sets whether the widget's label uses ' &' to indicate shortcuts.

• virtual void show ()
  Makes a widget visible.

• void size (int W, int H)
  Changes the size of the widget.

• int take_focus ()
  Gives the widget the keyboard focus.

• unsigned int takesevents () const
  Returns if the widget is able to take events.

• int test_shortcut ()
  Returns true if the widget's label contains the entered ' &x' shortcut.

• const char * tooltip () const
  Gets the current tooltip text.

• void tooltip (const char * text)
  Sets the current tooltip text.

• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const
  Gets the widget type.

• void type (uchar t)
  Sets the widget type.

• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void * user_data () const
  Gets the user data for this widget.

• void user_data (Fl_Callback_User_Data *v, bool auto_free)
  Sets the user data for this widget.

• void user_data (void *v)
  Sets the user data for this widget.

• unsigned int visible () const
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

• void visible_focus (int v)
  Modifies keyboard focus navigation.

• int visible_r () const
  Returns whether a widget and all its parents are visible.
• int w () const
  Gets the widget width.
• Fl__When when () const
  Returns the conditions under which the callback is called.
• void when (uchar i)
  Sets the flags used to decide when a callback is called.
• Fl__Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.
• int x () const
  Gets the widget position in its window.
• int y () const
  Gets the widget position in its window.
• virtual ~Fl__Widget ()
  Destroys the widget.

Static Public Member Functions

• static void default_callback (Fl__Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
• static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
• static int test_shortcut (const char *t, const bool requireAlt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types

• enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19, NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
flags possible values enumeration.

Protected Member Functions

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void draw_box () const
  Draws the widget box according its box style.
• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
• void draw_focus () const
**Draws a focus rectangle around the widget.**

- void `draw_focus` (Fl_Boxtype t, int X, int Y, int W, int H) const
  - Draws a focus rectangle around the widget.

- void `draw_focus` (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  - Draws a focus box for the widget at the given position and size.

- void `draw_label` () const
  - Draws the widget's label at the defined label position.

- void `draw_label` (int, int, int, int) const
  - Draws the label in an arbitrary bounding box.

- `Fl_Widget` (int x, int y, int w, int h, const char ∗label=0L)
  - Creates a widget at the given position and size.

- unsigned int `flags` () const
  - Gets the widget flags mask.

- void `h` (int v)
  - Internal use only.

- void `set_flag` (unsigned int c)
  - Sets a flag in the flags mask.

- void `w` (int v)
  - Internal use only.

- void `x` (int v)
  - Internal use only.

- void `y` (int v)
  - Internal use only.

---

**Friends**

- class `Fl_Group`

---

### 12.160.1 Detailed Description

`Fl_Widget` is the base class for all widgets in FLTK. You can't create one of these because the constructor is not public. However you can subclass it. All "property" accessing methods, such as `color()`, `parent()`, or `argument()` are implemented as trivial inline functions and thus are as fast and small as accessing fields in a structure. Unless otherwise noted, the property setting methods such as `color(n)` or `label(s)` are also trivial inline functions, even if they change the widget's appearance. It is up to the user code to call `redraw()` after these.

---

### 12.160.2 Member Enumeration Documentation

#### 12.160.2.1 anonymous enum

**anonymous enum** [protected]

flags possible values enumeration.

See `activate()`, `output()`, `visible()`, `changed()`, `set_visible_focus()`

**Enumerator**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INACTIVE</td>
<td>the widget can't receive focus, and is disabled but potentially visible</td>
</tr>
<tr>
<td>INVISIBLE</td>
<td>the widget is not drawn, but can receive a few special events</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>for output only</td>
</tr>
<tr>
<td>NOBORDER</td>
<td>don't draw a decoration (<code>Fl_Window</code>)</td>
</tr>
<tr>
<td>FORCE_POSITION</td>
<td>don't let the window manager position the window (<code>Fl_Window</code>)</td>
</tr>
<tr>
<td>NON_MODAL</td>
<td>this is a hovering toolbar window (<code>Fl_Window</code>)</td>
</tr>
<tr>
<td>SHORTCUT_LABEL</td>
<td>the label contains a shortcut we need to draw</td>
</tr>
</tbody>
</table>
### Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANGED</td>
<td>the widget value changed</td>
</tr>
<tr>
<td>OVERRIDE</td>
<td>position window on top (Fl_Window)</td>
</tr>
<tr>
<td>VISIBLE_FOCUS</td>
<td>accepts keyboard focus navigation if the widget can have the focus</td>
</tr>
<tr>
<td>COPIED_LABEL</td>
<td>the widget label is internally copied, its destruction is handled by the widget</td>
</tr>
<tr>
<td>CLIP_CHILDREN</td>
<td>all drawing within this widget will be clipped (Fl_Group)</td>
</tr>
<tr>
<td>MENU_WINDOW</td>
<td>a temporary popup window, dismissed by clicking outside (Fl_Window)</td>
</tr>
<tr>
<td>TOOLTIP_WINDOW</td>
<td>a temporary popup, transparent to events, and dismissed easily (Fl_Window)</td>
</tr>
<tr>
<td>MODAL</td>
<td>a window blocking input to all other windows (Fl_Window)</td>
</tr>
<tr>
<td>NO_OVERLAY</td>
<td>window not using a hardware overlay plane (Fl_Menu_Window)</td>
</tr>
<tr>
<td>GROUP_RELATIVE</td>
<td>Reserved, not implemented. DO NOT USE.</td>
</tr>
<tr>
<td>COPIED_TOOLTIP</td>
<td>the widget tooltip is internally copied, its destruction is handled by the widget</td>
</tr>
<tr>
<td>FULLSCREEN</td>
<td>a fullscreen window (Fl_Window)</td>
</tr>
<tr>
<td>MAC_USE_ACCENTS_MENU</td>
<td>On the macOS platform, pressing and holding a key on the keyboard opens an accented-character menu window (Fl_Input_, Fl_Text_Editor)</td>
</tr>
<tr>
<td>NEEDS_KEYBOARD</td>
<td>set on touch screen devices if a widget needs a keyboard when it gets the focus. Reserved, not yet used in 1.4.0. See also Fl_Widget::needs_keyboard()</td>
</tr>
<tr>
<td>IMAGE_BOUND</td>
<td>binding the image to the widget will transfer ownership, so that the widget will delete the image when it is no longer needed</td>
</tr>
<tr>
<td>DEIMAGE_BOUND</td>
<td>bind the inactive image to the widget, so the widget deletes the image when it is no longer needed</td>
</tr>
<tr>
<td>AUTO_DELETE_USER_DATA</td>
<td>automatically call delete on the user_data pointer when destroying this widget; if set, user_data must point to a class derived from the class Fl_Callback_User_Data</td>
</tr>
<tr>
<td>MAXIMIZED</td>
<td>a maximized Fl_Window</td>
</tr>
<tr>
<td>USERFLAG3</td>
<td>reserved for 3rd party extensions</td>
</tr>
<tr>
<td>USERFLAG2</td>
<td>reserved for 3rd party extensions</td>
</tr>
<tr>
<td>USERFLAG1</td>
<td>reserved for 3rd party extensions</td>
</tr>
</tbody>
</table>

### 12.160.3 Constructor & Destructor Documentation

#### 12.160.3.1 Fl_Widget()

```cpp
Fl_Widget::Fl_Widget ( int x, int y, int w, int h, const char ∗ label = 0L ) [protected]
```

Creates a widget at the given position and size.

The `Fl_Widget` is a protected constructor, but all derived widgets have a matching public constructor. It takes a value for `x()`, `y()`, `w()`, `h()`, and an optional value for `label()`.

**Parameters**

<table>
<thead>
<tr>
<th>in x,y</th>
<th>the position of the widget relative to the enclosing window</th>
</tr>
</thead>
<tbody>
<tr>
<td>in w,h</td>
<td>size of the widget in pixels</td>
</tr>
<tr>
<td>in label</td>
<td>optional text for the widget label</td>
</tr>
</tbody>
</table>
12.160.3.2 \texttt{\textasciitilde}FL\_Widget() 

\texttt{FL\_Widget::\textasciitilde\texttt{FL\_Widget} ( ) [virtual]}

Destroys the widget.

Destroys the widget, taking care of throwing focus before if any.

Destroying single widgets is not very common. You almost always want to destroy the parent group instead, which
will destroy all of the child widgets and groups in that group.

\begin{itemize}
\item Since FLTK 1.3, the widget's destructor removes the widget from its parent group, if it is member of a group.
\end{itemize}

Destruction removes the widget from any parent group! And groups when destroyed destroy all their children. This
is convenient and fast.

12.160.4 Member Function Documentation

12.160.4.1 activate() 

\texttt{void FL\_Widget::activate ( )}

Activates the widget.

Changing this value will send FL\_ACTIVATE to the widget if \texttt{active\_r()} is true.

See also \texttt{active(), active\_r(), deactivate()}

12.160.4.2 active() 

\texttt{unsigned int FL\_Widget::active ( ) const [inline]}

Returns whether the widget is active.

\begin{itemize}
\item Return values
\end{itemize}

\begin{itemize}
\item \texttt{0} if the widget is inactive
\end{itemize}

See also \texttt{active\_r(), activate(), deactivate()}

12.160.4.3 active\_r() 

\texttt{int FL\_Widget::active\_r ( ) const}

Returns whether the widget and all of its parents are active.

\begin{itemize}
\item Return values
\end{itemize}

\begin{itemize}
\item \texttt{0} if this or any of the parent widgets are inactive
\end{itemize}

See also \texttt{active(), activate(), deactivate()}

12.160.4.4 align() [1/2]

\texttt{FL\_Align FL\_Widget::align ( ) const [inline]}

Gets the label alignment.
Returns

label alignment

See also

label(), align(Fl_Align), Fl_Align

12.160.4.5 align() [2/2]

void Fl_Widget::align (Fl_Align alignment) [inline]

Sets the label alignment.
This controls how the label is displayed next to or inside the widget. The default value is FL_ALIGN_CENTER, which centers the label inside the widget.

Parameters

in alignment new label alignment

See also

align(), Fl_Align

12.160.4.6 argument() [1/2]

long Fl_Widget::argument ( ) const [inline]

Gets the current user data (long) argument that is passed to the callback function.

Note

On platforms with sizeof(long) < sizeof(void*), particularly on Windows 64-bit platforms, this method can truncate stored addresses (void*) to the size of a long value. Use with care and only if you are sure that the stored user_data value fits in a long value because it was stored with argument(long) or another method using only long values. You may want to use user_data() instead.

See also

user_data()

12.160.4.7 argument() [2/2]

void Fl_Widget::argument ( long v ) [inline]

Sets the current user data (long) argument that is passed to the callback function.

See also

argument()

12.160.4.8 as_gl_window()

virtual class Fl_Gl_Window * Fl_Widget::as_gl_window ( ) [inline], [virtual]

Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl Window.
Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Gl_Window. If it returns non-NULL, then the widget in question is derived from Fl_Gl_Window.
Return values

**NULL** if this widget is not derived from **Fl_Gl_Window**.

Note

This method is provided to avoid dynamic_cast.

See also

**Fl_Widget::as_group(), Fl_Widget::as_window()**

Reimplemented in **Fl_Gl_Window**.

### 12.160.4.9 as_group()

**virtual Fl_Group * Fl_Widget::as_group () [inline], [virtual]**

Returns an **Fl_Group** pointer if this widget is an **Fl_Group**. Use this method if you have a widget (pointer) and need to know whether this widget is derived from **Fl_Group**. If it returns non-NULL, then the widget in question is derived from **Fl_Group**, and you can use the returned pointer to access its children or other **Fl_Group**-specific methods.

Example:

```c
void my_callback (Fl_Widget *w, void *) {
    Fl_Group *g = w->as_group();
    if (g)
        printf ("This group has %d children\n", g->children());
    else
        printf ("This widget is not a group!\n");
}
```

Return values

**NULL** if this widget is not derived from **Fl_Group**.

Note

This method is provided to avoid dynamic_cast.

See also

**Fl_Widget::as_window(), Fl_Widget::as_gl_window()**

Reimplemented in **Fl_Group**.

### 12.160.4.10 as_window()

**virtual Fl_Window * Fl_Widget::as_window () [inline], [virtual]**

Returns an **Fl_Window** pointer if this widget is an **Fl_Window**. Use this method if you have a widget (pointer) and need to know whether this widget is derived from **Fl_Window**. If it returns non-NULL, then the widget in question is derived from **Fl_Window**, and you can use the returned pointer to access its children or other **Fl_Window**-specific methods.

Return values

**NULL** if this widget is not derived from **Fl_Window**.

Note

This method is provided to avoid dynamic_cast.
See also

Fl_Widget::as_group(), Fl_Widget::as_gl_window()

Reimplemented in Fl_Window.

### 12.160.4.11 bind_deimage() [1/2]

```cpp
void Fl_Widget::bind_deimage (Fl_Image *img)
```

Sets the image to use as part of the widget label when in the inactive state.

**Parameters**

| in | img | the new image for the deactivated widget |

**Note**

The image will be bound to the widget. When the widget is deleted, the image will be deleted as well.

See also

void deimage(Fl_Image* img)

### 12.160.4.12 bind_deimage() [2/2]

```cpp
void Fl_Widget::bind_deimage (int f) [inline]
```

Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

**Parameters**

| f | 1: mark the image as bound, 0: mark the image as managed by the user |

See also

deimage_bound(), bind_image()

### 12.160.4.13 bind_image() [1/2]

```cpp
void Fl_Widget::bind_image (Fl_Image *img)
```

Sets the image to use as part of the widget label when in the active state.

The image will be bound to the widget. When the widget is deleted, the image will be deleted as well.

Calling bind_image() with a new image will delete the old image if it was bound, and then set the new image, and bind that. If old and new image are the same, nothing happens.

Calling bind_image() with NULL will delete the old image if it was bound and not set a new image.

**Parameters**

| in | img | the new image for the label |
See also

```cpp
void image(Fl_Imagew* img)
```

### 12.160.4.14 bind_image() [2/2]

```cpp
void Fl_Widget::bind_image ( int f ) [inline]
```

Bind the image to the widget, so the widget will delete the image when it is no longer needed.

**Parameters**

| f | 1: mark the image as bound, 0: mark the image as managed by the user |

See also

```cpp
image_bound(), bind_deimage()
```

### 12.160.4.15 box() [1/2]

```cpp
Fl_Boxtype Fl_Widget::box ( ) const [inline]
```

Gets the box type of the widget.

**Returns**

the current box type

See also

```cpp
box(Fl_Boxtype), Fl_Boxtype
```

### 12.160.4.16 box() [2/2]

```cpp
void Fl_Widget::box ( Fl_Boxtype new_box ) [inline]
```

Sets the box type for the widget.

This identifies a routine that draws the background of the widget. See Fl_Boxtype for the available types. The default depends on the widget, but is usually FL_NO_BOX or FL_UP_BOX.

**Parameters**

| new_box | the new box type |

See also

```cpp
box(), Fl_Boxtype
```

### 12.160.4.17 callback() [1/6]

```cpp
Fl_Callback_p Fl_Widget::callback ( ) const [inline]
```

Gets the current callback function for the widget.

Each widget has a single callback.

**Returns**

current callback
12.160.4.18 callback() [2/6]

void Fl_Widget::callback (  
    Fl_Callback * cb ) [inline]

Sets the current callback function for the widget.  
Each widget has a single callback.

Parameters

| in  | cb     | new callback |

12.160.4.19 callback() [3/6]

void Fl_Widget::callback (  
    Fl_Callback * cb,  
    Fl_Callback_User_Data * p,  
    bool auto_free ) [inline]

Sets the current callback function and managed user data for the widget.  
Setting auto_free will transfer ownership of the callback user data to the widget. Deleting the widget will then also delete the user data.

Parameters

| in  | cb     | new callback |
| in  | p      | user data    |
| in  | auto_free | if set, the widget will free user data when destroyed |

12.160.4.20 callback() [4/6]

void Fl_Widget::callback (  
    Fl_Callback * cb,  
    void * p ) [inline]

Sets the current callback function and data for the widget.  
Each widget has a single callback.

Parameters

| in  | cb     | new callback |
| in  | p      | user data    |

12.160.4.21 callback() [5/6]

void Fl_Widget::callback (  
    Fl_Callback0 * cb ) [inline]

Sets the current callback function for the widget.  
Each widget has a single callback.

Parameters

| in  | cb     | new callback |

12.160.4.22 callback() [6/6]

void Fl_Widget::callback (  

Generated by Doxygen
Sets the current callback function for the widget.
Each widget has a single callback.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cb</td>
<td>Fl_Callback</td>
<td>new callback</td>
</tr>
<tr>
<td>p</td>
<td>long</td>
<td>user data</td>
</tr>
</tbody>
</table>

### 12.160.4.23  changed()

```cpp
unsigned int Fl_Widget::changed ( ) const [inline]
```

Checks if the widget value changed since the last callback.

"Changed" is a flag that is turned on when the user changes the value stored in the widget. This is only used by subclasses of `Fl_Widget` that store values, but is in the base class so it is easier to scan all the widgets in a panel and `do_callback()` on the changed ones in response to an "OK" button.

Most widgets turn this flag off when they do the callback, and when the program sets the stored value.

**Note**

`do_callback()` turns this flag off after the callback.

**Return values**

- `0` if the value did not change

**See also**

- `set_changed()`, `clear_changed()`
- `do_callback(Fl_Widget *widget, void *data)`

### 12.160.4.24  clear_active()

```cpp
void Fl_Widget::clear_active ( ) [inline]
```

Marks the widget as inactive without sending events or changing focus.
This is mainly for specialized use, for normal cases you want `deactivate()`.

**See also**

- `deactivate()`

### 12.160.4.25  clear_changed()

```cpp
void Fl_Widget::clear_changed ( ) [inline]
```

Marks the value of the widget as unchanged.

**See also**

- `changed()`, `set_changed()`

### 12.160.4.26  clear_damage()

```cpp
void Fl_Widget::clear_damage ( uchar c = 0 ) [inline]
```

Clears or sets the damage flags.
Damage flags are cleared when parts of the widget drawing is repaired.
The optional argument `c` specifies the bits that are set after the call (default: 0) and not the bits that are cleared!
Therefore it is possible to set damage bits with this method, but this should be avoided. Use `damage(uchar)` instead.

**Parameters**

| in  | c | new bitmask of damage flags (default: 0) |

See also

`damage(uchar), damage()`

### 12.160.4.27 clear_output()

```cpp
void Fl_Widget::clear_output () [inline]
```

Sets a widget to accept input.

See also

`set_output(), output()`

### 12.160.4.28 clear_visible()

```cpp
void Fl_Widget::clear_visible () [inline]
```

Hides the widget.

You must still redraw the parent to see a change in the window. Normally you want to use the `hide()` method instead.

### 12.160.4.29 clear_visible_focus()

```cpp
void Fl_Widget::clear_visible_focus () [inline]
```

Disables keyboard focus navigation with this widget.

Normally, all widgets participate in keyboard focus navigation.

See also

`set_visible_focus(), visible_focus(), visible_focus(int)`

### 12.160.4.30 color () [1/3]

```cpp
Fl_Color Fl_Widget::color ( void ) const [inline]
```

 Gets the background color of the widget.

**Returns**

current background color

See also

`color(Fl_Color), color(Fl_Color, Fl_Color)`

### 12.160.4.31 color () [2/3]

```cpp
void Fl_Widget::color ( Fl_Color bg ) [inline]
```

Sets the background color of the widget.

The color is passed to the box routine. The color is either an index into an internal table of RGB colors or an RGB color value generated using `fl_rgb_color()`.

The default for most widgets is FL_BACKGROUND_COLOR. Use `Fl::set_color()` to redefine colors in the color map.
Parameters

- **in** `bg` background color

See also

- `color()`, `color(Fl_Color, Fl_Color)`, `selection_color(Fl_Color)`

### 12.160.4.32 color()

```cpp
void Fl_Widget::color (Fl_Color bg, Fl_Color sel) [inline]
```

Sets the background and selection color of the widget. The two color form sets both the background and selection colors.

Parameters

- **in** `bg` background color
- **in** `sel` selection color

See also

- `color(unsigned)`, `selection_color(unsigned)`

### 12.160.4.33 color2()

```cpp
Fl_Color Fl_Widget::color2 () const [inline]
```

For back compatibility only.

**Deprecated** Use `selection_color()` instead.

### 12.160.4.34 color2()

```cpp
void Fl_Widget::color2 (unsigned a) [inline]
```

For back compatibility only.

**Deprecated** Use `selection_color(unsigned)` instead.

### 12.160.4.35 contains()

```cpp
int Fl_Widget::contains (const Fl_Widget * w) const
```

Checks if `w` is a child of this widget.

Parameters

- **in** `w` potential child widget

Returns

- Returns 1 if `w` is a child of this widget, or is equal to this widget. Returns 0 if `w` is NULL.
12.160.4.36  copy_label()

```cpp
void Fl_Widget::copy_label ( const char * new_label )
```

Sets the current label.
Unlike `label()`, this method allocates a copy of the label string instead of using the original string pointer. The internal copy will automatically be freed whenever you assign a new label or when the widget is destroyed.

**Parameters**

- `new_label` the new label text

**See also**

- `label()`

12.160.4.37  copy_tooltip()

```cpp
void Fl_Widget::copy_tooltip ( const char * text )
```

Sets the current tooltip text.
 Unlike `tooltip()`, this method allocates a copy of the tooltip string instead of using the original string pointer. The internal copy will automatically be freed whenever you assign a new tooltip or when the widget is destroyed.
If no tooltip is set, the tooltip of the parent is inherited. Setting a tooltip for a group and setting no tooltip for a child will show the group's tooltip instead. To avoid this behavior, you can set the child's tooltip to an empty string ("").

**Parameters**

- `text` New tooltip text (an internal copy is made and managed)

**See also**

- `tooltip(const char*)`, `tooltip()`

12.160.4.38  damage() [1/3]

```cpp
uchar Fl_Widget::damage ( ) const [inline]
```

Returns non-zero if `draw()` needs to be called.
The damage value is actually a bit field that the widget subclass can use to figure out what parts to draw.

**Returns**

- a bitmap of flags describing the kind of damage to the widget

**See also**

- `damage(uchar)`, `clear_damage(uchar)`

12.160.4.39  damage() [2/3]

```cpp
void Fl_Widget::damage ( uchar c )
```

Sets the damage bits for the widget.
Setting damage bits will schedule the widget for the next redraw.

**Parameters**

- `c` bitmask of flags to set
See also

\texttt{damage()}, \texttt{clear\_damage(uchar)}

\subsection*{12.160.4.40 \texttt{damage}() [3/3]}

\texttt{void Fl\_Widget::damage (}
\texttt{
\texttt{uchar c,}
\texttt{int x,}
\texttt{int y,}
\texttt{int w,}
\texttt{int h )}
\texttt{)}

Sets the damage bits for an area inside the widget.
Setting damage bits will schedule the widget for the next redraw.

\begin{tabular}{|l|l|}
\hline
\textbf{in} & \texttt{c} & bitmask of flags to set \\
\hline
\textbf{in} & \texttt{x,y,w,h} & size of damaged area \\
\hline
\end{tabular}

See also

\texttt{damage()}, \texttt{clear\_damage(uchar)}

\subsection*{12.160.4.41 \texttt{deactivate}()}

\texttt{void Fl\_Widget::deactivate ( )}
Deactivates the widget.
Inactive widgets will be drawn "grayed out", e.g. with less contrast than the active widget. Inactive widgets will
not receive any keyboard or mouse button events. Other events (including FL\_ENTER, FL\_MOVE, FL\_LEAVE,
FL\_SHORTCUT, and others) will still be sent. A widget is only active if \texttt{active()} is true on it \textit{and all of its parents}.
Changing this value will send FL\_DEACTIVATE to the widget if \texttt{active\_r()} is true.
Currently you cannot deactivate \texttt{Fl\_Window} widgets.
See also

\texttt{activate()}, \texttt{active()}, \texttt{active\_r()}

\subsection*{12.160.4.42 \texttt{default\_callback}()}

\texttt{void Fl\_Widget::default\_callback (}
\texttt{
\texttt{Fl\_Widget \texttt{*} widget,}
\texttt{void \texttt{*} data ) \ [static]}
\texttt{)}

The \texttt{default\_callback} for all widgets that don't set a callback.
This callback function puts a pointer to the widget on the queue returned by \texttt{Fl\_readqueue()}. This is the default for
all widgets if you don't set a callback.
You can avoid the overhead of this default handling if you set the callback to \texttt{NULL} explicitly.
Relying on the default callback and reading the callback queue with \texttt{Fl\_readqueue()} is not recommended. If you
need a callback, you should set one with \texttt{Fl\_Widget::callback(Fl\_Callback cb, void \texttt{*} data)} or one of its variants.

\begin{tabular}{|l|l|}
\hline
\textbf{in} & \texttt{widget} & the \texttt{Fl\_Widget} given to the callback \\
\hline
\textbf{in} & \texttt{data} & user data associated with that callback \\
\hline
\end{tabular}
See also

callback(), Fl::readqueue()
do_callback(Fl_Widget *widget, void *data)

12.160.4.43 deimage() [1/4]

Fl_Image * Fl_Widget::deimage () [inline]
Gets the image that is used as part of the widget label when in the inactive state.

Returns

the current image for the deactivated widget

12.160.4.44 deimage() [2/4]

const Fl_Image * Fl_Widget::deimage () const [inline]
Gets the image that is used as part of the widget label when in the inactive state.

Returns

the current image for the deactivated widget

12.160.4.45 deimage() [3/4]

void Fl_Widget::deimage ( Fl_Image & img )
Sets the image to use as part of the widget label when in the inactive state.

Parameters

| in  | img  | the new image for the deactivated widget |

See also

void deimage(Fl_Image* img)

12.160.4.46 deimage() [4/4]

void Fl_Widget::deimage ( Fl_Image * img )
Sets the image to use as part of the widget label when in the inactive state.

Parameters

| in  | img  | the new image for the deactivated widget |

Note

The caller is responsible for making sure img is not deleted while it's used by the widget, and, if appropriate, for deleting it after the widget's deletion.

See also

void bind_deimage(Fl_Image* img)
12.160.4.47 deimage_bound()

```cpp
int Fl_Widget::deimage_bound ( ) const [inline]
```

Returns whether the inactive image is managed by the widget.

### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The image is not bound to the widget</td>
</tr>
<tr>
<td>1</td>
<td>The image will be deleted when the widget is deleted</td>
</tr>
</tbody>
</table>

See also

- `image_bound()`, `bind_deimage()`

12.160.4.48 do_callback () [1/3]

```cpp
void Fl_Widget::do_callback ( Fl_Callback_Reason reason = FL_REASON_UNKNOWN ) [inline]
```

Calls the widget callback function with default arguments.

This is the same as calling

```cpp
do_callback(this, user_data(), reason);
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reason</code></td>
<td>Give a reason to why this callback was called, defaults to <code>FL_REASON_UNKNOWN</code></td>
</tr>
</tbody>
</table>

See also

- `callback()`
  - `do_callback(Fl_Widget *widget, void *data, Fl_Callback_Reason reason), Fl_Callback_Reason`

12.160.4.49 do_callback () [2/3]

```cpp
void Fl_Widget::do_callback ( Fl_Widget * widget, long arg,
                            Fl_Callback_Reason reason = FL_REASON_UNKNOWN ) [inline]
```

Calls the widget callback function with arbitrary arguments.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>widget</code></td>
<td>Call the callback with <code>widget</code> as the first argument</td>
</tr>
<tr>
<td><code>arg</code></td>
<td>Call the callback with <code>arg</code> as the user data (second) argument</td>
</tr>
<tr>
<td><code>reason</code></td>
<td>Give a reason to why this callback was called, defaults to <code>FL_REASON_UNKNOWN</code></td>
</tr>
</tbody>
</table>

See also

- `callback()`
  - `do_callback(Fl_Widget *widget, void *data), Fl_Callback_Reason`

12.160.4.50 do_callback () [3/3]

```cpp
void Fl_Widget::do_callback ( Fl_Widget * widget, void * arg = 0,
                            Fl_Callback_Reason reason = FL_REASON_UNKNOWN )
```

Generated by Doxygen
Calls the widget callback function with arbitrary arguments.
All overloads of do_callback() call this method. It does nothing if the widget's callback() is NULL. It clears the widget's changed flag after the callback was called unless the callback is the default callback. Hence it is not necessary to call clear_changed() after calling do_callback() in your own widget's handle() method.

A reason must be set for widgets if different actions can trigger the same callback.

Note
It is legal to delete the widget in the callback (i.e. in user code), but you must not access the widget in the handle() method after calling do_callback() if the widget was deleted in the callback. We recommend to use Fl_Widget_Tracker to check whether the widget was deleted in the callback.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>widget</th>
<th>call the callback with widget as the first argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>arg</td>
<td>use arg as the user data (second) argument</td>
</tr>
<tr>
<td>in</td>
<td>reason</td>
<td>give a reason to why this callback was called, defaults to FL_REASON_UNKNOWN</td>
</tr>
</tbody>
</table>

See also
- default_callback()
- callback()
- class Fl_Widget_Tracker
- Fl::callback_reason()

12.160.4.51 draw()

virtual void Fl_Widget::draw ( ) [pure virtual]

Draws the widget. Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget’s draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:
```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

12.160.4.52 draw_focus() [1/3]

void Fl_Widget::draw_focus ( ) const [inline], [protected]

Draws a focus rectangle around the widget.
This method uses the widget’s boxtype and coordinates and its background color color().

See also
- Fl_Widget::draw_focus(Fl_Boxtype, int, int, int, Fl_Color) const

12.160.4.53 draw_focus() [2/3]

void Fl_Widget::draw_focus ( Fl_Boxtype t,
Draws a focus rectangle around the widget.
This method uses the given boxtype and coordinates and the widget's background color color().

See also

`Fl_Widget::draw_focus(Fl_Boxtype, int, int, int, int, Fl_Color) const`

12.160.4.54 draw_focus() [3/3]

```cpp
void Fl_Widget::draw_focus (Fl_Boxtype bt, int X, int Y, int W, int H, Fl_Color bg) const [protected]
```

Draws a focus box for the widget at the given position and size.
This method does nothing if

- the global option Fl::visible_focus() or
- the per-widget option visible_focus() is false (off).

This means that Fl_Widget::draw_focus() or one of the more specialized methods can be called without checking these visible focus options.

Note

This method must only be called if the widget has the focus. This is not tested internally.

The boxtype `bt` is used to calculate the inset so the focus box is drawn inside the box borders.
The default focus box drawing color is black. The background color `bg` is used to determine a better visible color if necessary by using fl_contrast() with the given background color.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>bt</th>
<th>Boxtyle that needs to be considered (frame width)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y,W,H</td>
<td>Bounding box</td>
</tr>
<tr>
<td>in</td>
<td>bg</td>
<td>Background color</td>
</tr>
</tbody>
</table>

See also

`Fl_Widget::draw_focus()`

`Fl_Widget::draw_focus(Fl_Boxtype, int, int, int, int) const`

12.160.4.55 draw_label() [1/3]

```cpp
void Fl_Widget::draw_label () const [protected]
```

Draws the widget's label at the defined label position.
This is the normal call for a widget's draw() method.

12.160.4.56 draw_label() [2/3]

```cpp
void Fl_Widget::draw_label (int X,
```
12.160 Fl_Widget Class Reference

int Y,
int W,
int H) const [protected]

Draws the label in an arbitrary bounding box.
draw()) can use this instead of draw_label(void) to change the bounding box

12.160.4.57 draw_label() [3/3]

void Fl_Widget::draw_label {
    int X,
    int Y,
    int W,
    int H,
    Fl_Align a ) const

Draws the label in an arbitrary bounding box with an arbitrary alignment.
Anybody can call this to force the label to draw anywhere.

12.160.4.58 h() [1/2]

int Fl_Widget::h ( ) const [inline]

Gets the widget height.

Returns

the height of the widget in pixels.

12.160.4.59 h() [2/2]

void Fl_Widget::h {
    int v ) [inline], [protected]

Internal use only.
Use position(int,int), size(int,int) or resize(int,int,int,int) instead.

12.160.4.60 handle()

int Fl_Widget::handle {
    int event ) [virtual]

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-
circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class.
This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The
effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses)
in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In
such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by
returning 1 even if you didn't do anything with it.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>event</th>
<th>the kind of event received</th>
</tr>
</thead>
</table>

Return values

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>if the event was not used or understood</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>
See also

Fi_Event

Reimplemented in Fl_Color_Chooser, Fl_Free, Fl_Table, Fl_Terminal, Fl_Text_Display, Fl_Text_Editor, Fl_Tree,
Fl_Browser, Fl_File_Input, Fl_Scheme_Choice, Fl_Spinner::Fl_Spinner_Input, Fl_Spinner, Fl_Table_Row,
Fl_Tile, Fl_Adjuster, Fl_Box, Fl_Button, Fl_Check_Browser, Fl_Check_Button, Fl_Clock, Fl_Count, Fl_Dial,
Fl_Gl_Window, Fl_Group, Fl_Help_View, Fl_Input, Fl_Light_Button, Fl_Menu_Bar, Fl_Menu_Button, Fl_Positioner,
Fl_Repeat_Button, Fl_Return_Button, Fl_Roller, Fl_Scroll, Fl_Scrollbar, Fl_Secret_Input, Fl_Shortcut_Button,
Fl_Slider, Fl_Tabs, Fl_Timer, Fl_Value_Input, Fl_Value_Output, Fl_Value_Slider, Fl_Window, and Fl_Glut_Window.

### 12.160.4.61 hide()

```cpp
void Fl_Widget::hide ( ) [virtual]
```

Makes a widget invisible.

See also

- show(), visible(), visible_r()

Reimplemented in Fl_Browser, Fl_Double_Window, Fl_Gl_Window, Fl_Overlay_Window, and Fl_Window.

### 12.160.4.62 image() [1/4]

```cpp
Fl_Image * Fl_Widget::image ( ) [inline]
```

Gets the image that is used as part of the widget label when in the active state.

Returns

the current image

### 12.160.4.63 image() [2/4]

```cpp
const Fl_Image * Fl_Widget::image ( ) const [inline]
```

Gets the image that is used as part of the widget label when in the active state.

Returns

the current image

### 12.160.4.64 image() [3/4]

```cpp
void Fl_Widget::image ( Fl_Image & img )
```

Sets the image to use as part of the widget label when in the active state.

Parameters

- **in img** the new image for the label

See also

- void image(Fl_Image* img)

### 12.160.4.65 image() [4/4]

```cpp
void Fl_Widget::image ( Fl_Image * img )
```

Sets the image to use as part of the widget label when in the active state.

The caller is responsible for making sure `img` is not deleted while it's used by the widget, and, if appropriate, for deleting it after the widget's deletion.
Calling \texttt{image()} with a new image will delete the old image if it was bound, and set the new image without binding it. If old and new are the same, the image will not be deleted, but it will be unbound.

Calling \texttt{image()} with NULL will delete the old image if it was bound and not set a new image.

\begin{verbatim}
Parameters
\begin{tabular}{l|l}
in & \texttt{img} \quad \text{the new image for the label}
\end{tabular}
\end{verbatim}

See also
\begin{verbatim}
bind_image(Fl(Image) img)
\end{verbatim}

\textbf{12.160.4.66 image_bound()}

\begin{verbatim}
int Fl_Widget::image_bound ( ) const [inline]
\end{verbatim}

Returns whether the image is managed by the widget.

\begin{verbatim}
Return values
\begin{tabular}{l|l}
0 & if the image is not bound to the widget
1 & if the image will be deleted when the widget is deleted
\end{tabular}
\end{verbatim}

See also
\begin{verbatim}
deimage_bound(), bind_image()
\end{verbatim}

\textbf{12.160.4.67 inside()}

\begin{verbatim}
int Fl_Widget::inside ( const Fl_Widget * wgt ) const [inline]
\end{verbatim}

Checks if this widget is a child of \texttt{wgt}.

Returns 1 if this widget is a child of \texttt{wgt}, or is equal to \texttt{wgt}. Returns 0 if \texttt{wgt} is NULL.

\begin{verbatim}
Parameters
\begin{tabular}{l|l}
in & \texttt{wgt} \quad \text{the possible parent widget}
\end{tabular}
\end{verbatim}

See also
\begin{verbatim}
contains()
\end{verbatim}

\textbf{12.160.4.68 is_label_copied()}

\begin{verbatim}
int Fl_Widget::is_label_copied ( ) const [inline]
\end{verbatim}

Returns whether the current label was assigned with \texttt{copy_label()}.

This can be useful for temporarily overwriting the widget's label and restoring it later.

\begin{verbatim}
Return values
\begin{tabular}{l|l}
0 & current label was assigned with \texttt{label()}. \\
1 & current label was assigned with \texttt{copy_label()}. \\
\end{tabular}
\end{verbatim}
12.160.4.69 label() [1/3]

const char * Fl_Widget::label() const [inline]

Gets the current label text.

Returns

a pointer to the current label text

See also

label(const char *), copy_label(const char *)

12.160.4.70 label() [2/3]

void Fl_Widget::label(const char * text)

Sets the current label pointer.
The label is shown somewhere on or next to the widget. See Labels and Label Types for details about what can be put in a label. The passed pointer is stored unchanged in the widget (the string is not copied), so if you need to set the label to a formatted value, make sure the buffer is static, global, or allocated. The copy_label() method can be used to make a copy of the label string automatically.

Parameters

\textbf{in} \hspace{1em} text \hspace{1em} pointer to new label text

See also

copy_label()

12.160.4.71 label() [3/3]

void Fl_Widget::label(Fl_Labeltype a, const char * b) [inline]

Shortcut to set the label text and type in one call.

See also

label(const char *), labeltype(Fl_Labeltype)

12.160.4.72 label_shortcut()

unsigned int Fl_Widget::label_shortcut(const char * t) [static]

Returns the Unicode value of the '&x' shortcut in a given text.
The given text \texttt{t} (usually a widget's label or a menu text) is searched for a '&x' shortcut label, and if found, the Unicode value (code point) of the '&x' shortcut is returned.

Parameters

\textbf{f} \hspace{1em} text or label to search for '&x' shortcut.

Returns

Unicode (UCS-4) value of shortcut in \texttt{t} or 0.
Note

Internal use only.

12.160.4.73  labelcolor() [1/2]

```cpp
Fl_Color Fl_Widget::labelcolor () const [inline]
```

Gets the label color.
The default color is FL_FOREGROUND_COLOR.

Returns

the current label color

12.160.4.74  labelcolor() [2/2]

```cpp
void Fl_Widget::labelcolor ( Fl_Color c ) [inline]
```

Sets the label color.
The default color is FL_FOREGROUND_COLOR.

Parameters

| in  | c          | the new label color |

12.160.4.75  labelfont() [1/2]

```cpp
Fl_Font Fl_Widget::labelfont () const [inline]
```

Gets the font to use.
Fonts are identified by indexes into a table. The default value uses a Helvetica typeface (Arial for Microsoft® Windows®). The function Fl::set_font() can define new typefaces.

Returns

current font used by the label

See also

Fl_Font

12.160.4.76  labelfont() [2/2]

```cpp
void Fl_Widget::labelfont ( Fl_Font f ) [inline]
```

Sets the font to use.
Fonts are identified by indexes into a table. The default value uses a Helvetica typeface (Arial for Microsoft® Windows®). The function Fl::set_font() can define new typefaces.

Parameters

| in  | f          | the new font for the label |
12.160.4.77 labelsize() [1/2]

`Fl_Fontsize Fl_Widget::labelsize ( ) const [inline]`

Gets the font size in pixels.
The default size is 14 pixels.

Returns

the current font size

12.160.4.78 labelsize() [2/2]

`void Fl_Widget::labelsize ( Fl_Fontsize pix ) [inline]`

Sets the font size in pixels.

Parameters

in pix the new font size

See also

`Fl_Fontsize labelsize()`

12.160.4.79 labeltype() [1/2]

`Fl_Labeltype Fl_Widget::labeltype ( ) const [inline]`

Gets the label type.

Returns

the current label type.

See also

`Fl_Labeltype`

12.160.4.80 labeltype() [2/2]

`void Fl_Widget::labeltype ( Fl_Labeltype a ) [inline]`

Sets the label type.
The label type identifies the function that draws the label of the widget. This is generally used for special effects such as embossing or for using the label() pointer as another form of data such as an icon. The value `FL_NORMAL_LABEL` prints the label as plain text.

Parameters

in a new label type

See also

`Fl_Labeltype`
### 12.160.4.81 measure_label()

```cpp
void Fl_Widget::measure_label (  
    int & ww,  
    int & hh ) const [inline]
```

Sets width `ww` and height `hh` accordingly with the label size. Labels with images will return `w()` and `h()` of the image.

This calls `fl_measure()` internally. For more information about the arguments `ww` and `hh` and word wrapping, see also:

- `fl_measure(const char*, int&, int&, int)`

### 12.160.4.82 needs_keyboard() [1/2]

```cpp
bool Fl_Widget::needs_keyboard ( ) const [inline]
```

Returns whether this widget needs a keyboard.

Returns

- `true` or `false`

See also

- `needs_keyboard(bool)`

### 12.160.4.83 needs_keyboard() [2/2]

```cpp
void Fl_Widget::needs_keyboard (  
    bool needs ) [inline]
```

Sets whether this widget needs a keyboard. Set this on touch screen devices if a widget needs a keyboard when it gets the focus.

**Note**

This flag can be set but is not yet **used** in FLTK 1.4.0. It is intended to be used in the future on real touch devices.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>needs</th>
<th>Set this to true or false</th>
</tr>
</thead>
</table>

### 12.160.4.84 output()

```cpp
unsigned int Fl_Widget::output ( ) const [inline]
```

Returns if a widget is used for output only.

**output()** means the same as `!active()` except it does not change how the widget is drawn. The widget will not receive any events. This is useful for making scrollbars or buttons that work as displays rather than input devices.

**Return values**

| 0   | if the widget is used for input and output |
12.160.4.85  parent() [1/2]

Fl_Group * Fl_Widget::parent () const [inline]

Returns a pointer to the parent widget.
Usually this is a Fl_Group or Fl_Window.

Return values

NULL if the widget has no parent

See also

Fl_Group::add(Fl_Widget+)

12.160.4.86  parent() [2/2]

void Fl_Widget::parent ( Fl_Group * p ) [inline]

Internal use only - "for hacks only".
It is STRONGLY recommended not to use this method, because it short-circuits Fl_Group's normal widget adding and removing methods, if the widget is already a child widget of another Fl_Group.
Use Fl_Group::add(Fl_Widget+) and/or Fl_Group::remove(Fl_Widget+) instead.

12.160.4.87  position()

void Fl_Widget::position ( int X, int Y ) [inline]

Repositions the window or widget.
position(X, Y) is a shortcut for resize(X, Y, w(), h()).

Parameters

in X,Y new position relative to the parent window

See also

resize(int,int,int,int), size(int,int)

12.160.4.88  redraw()

void Fl_Widget::redraw ( )

Schedules the drawing of the widget.
Marks the widget as needing its draw() routine called.

12.160.4.89  redraw_label()

void Fl_Widget::redraw_label ( )

Schedules the drawing of the label.
Marks the widget or the parent as needing a redraw for the label area of a widget.

12.160.4.90  resize()

void Fl_Widget::resize ( 

See also

set_output(), clear_output()
Changes the size or position of the widget.
This is a virtual function so that the widget may implement its own handling of resizing. The default version does not call the `redraw()` method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position.
Some window managers under X11 call `resize()` a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations.
`position(X, Y)` is a shortcut for `resize(X, Y, w(), h())`, and `size(W, H)` is a shortcut for `resize(x(), y(), W, H)`.

### Parameters

| in | x, y | new position relative to the parent window |
|    | w, h | new size |

See also

- `position(int, int)`, `size(int, int)`


### 12.160.4.91 selection_color() [1/2]

#### Fl_Color Fl_Widget::selection_color ( ) const [inline]

Gets the selection color.

Returns

the current selection color

See also

- `selection_color(Fl_Color)`, `color(Fl_Color, Fl_Color)`

### 12.160.4.92 selection_color() [2/2]

#### void Fl_Widget::selection_color ( Fl_Color a ) [inline]

Sets the selection color.
The selection color is defined for Forms compatibility and is usually used to color the widget when it is selected, although some widgets use this color for other purposes. You can set both colors at once with `color(Fl_Color bg, Fl_Color sel)`.

Parameters

| in | a | the new selection color |

See also

- `selection_color()`, `color(Fl_Color, Fl_Color)`

### 12.160.4.93 set_active()

#### void Fl_Widget::set_active ( ) [inline]

Generated by Doxygen
Marks the widget as active without sending events or changing focus. This is mainly for specialized use, for normal cases you want `activate()`.

See also `activate()`

### 12.160.4.94 set_changed()
```cpp
void Fl_Widget::set_changed ( ) [inline]
```
Marks the value of the widget as changed.

See also `changed()`, `clear_changed()`

### 12.160.4.95 set_output()
```cpp
void Fl_Widget::set_output ( ) [inline]
```
Sets a widget to output only.

See also `output()`, `clear_output()`

### 12.160.4.96 set_visible()
```cpp
void Fl_Widget::set_visible ( ) [inline]
```
Makes the widget visible.

You must still redraw the parent widget to see a change in the window. Normally you want to use the `show()` method instead.

### 12.160.4.97 set_visible_focus()
```cpp
void Fl_Widget::set_visible_focus ( ) [inline]
```
Enables keyboard focus navigation with this widget.

Note, however, that this will not necessarily mean that the widget will accept focus, but for widgets that can accept focus, this method enables it if it has been disabled.

See also `visible_focus()`, `clear_visible_focus()`, `visible_focus(int)`

### 12.160.4.98 shortcut_label() [1/2]
```cpp
int Fl_Widget::shortcut_label ( ) const [inline]
```
Returns whether the widget's label uses `&` to indicate shortcuts.

See also `void shortcut_label(int value)`

### 12.160.4.99 shortcut_label() [2/2]
```cpp
void Fl_Widget::shortcut_label ( int value ) [inline]
```
Sets whether the widget's label uses `&` to indicate shortcuts.

By default, all objects of classes `Fl_Menu_` (and derivatives), `Fl_Button` (and derivatives), `Fl_Text_Display`, `Fl_Value_Input`, and `Fl_Input_` (and derivatives) use character `&` in their label, unless `&` is repeated, to indicate shortcuts: `&` does not appear in the drawn label, the next character after `&` in the label is drawn underlined, and typing this character triggers the corresponding menu window, button, or other widget. If the label contains 2 consecutive `&`, only one is drawn and the next character is not underlined and not used as a shortcut. If `value` is set to 0, all these labels don't process character `&` as indicating a shortcut: `&` is drawn in the label, the next character is not underlined and does not define a shortcut.
12.160.4.100  show()

void Fl_Widget::show ( ) [virtual]

Makes a widget visible.

An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other
events like FL_SHOW.

The visible() method returns true if the widget is set to be visible. The visible_r() method returns true if the widget
and all of its parents are visible. A widget is only visible if visible() is true on it and all of its parents.

Changing it will send FL_SHOW or FL_HIDE events to the widget. Do not change it if the parent is not visible, as
this will send false FL_SHOW or FL_HIDE events to the widget. redraw() is called if necessary on this or the parent.

See also

hide(), visible(), visible_r()

Reimplemented in Fl_Browser, Fl_Double_Window, Fl_Gl_Window, Fl_Overlay_Window, Fl_Single_Window, and
Fl_Window.

12.160.4.101  size()

void Fl_Widget::size (  
    int W,
    int H ) [inline]

Changes the size of the widget.

size(W, H) is a shortcut for resize(x(), y(), W, H).

Parameters

| in | W/H | new size |

See also

position(int,int), resize(int,int,int,int)

12.160.4.102  take_focus()

int Fl_Widget::take_focus ( )

Gives the widget the keyboard focus.

Tries to make this widget be the Fl::focus() widget, by first sending it an FL_FOCUS event, and if it returns non-zero,
setting Fl::focus() to this widget. You should use this method to assign the focus to a widget.

Returns

true if the widget accepted the focus.

12.160.4.103  takesevents()

unsigned int Fl_Widget::takesevents ( ) const [inline]

Returns if the widget is able to take events.

This is the same as (active() && !output() && visible()) but is faster.

Return values

| 0 | if the widget takes no events |

12.160.4.104  test_shortcut() [1/2]

int Fl_Widget::test_shortcut ( )
Returns true if the widget's label contains the entered '&x' shortcut. This method must only be called in handle() methods or callbacks after a keypress event (usually FL_KEYDOWN or FL_SHORTCUT). The widget's label is searched for a '&x' shortcut, and if found, this is compared with the entered key value. Fl::event_text() is used to get the entered key value.

Note
Useful when a widget's handle(int) method needs dedicated processing of FL_SHORTCUT.

12.160.4.105 test_shortcut() [2/2]

int Fl_Widget::test_shortcut (const char *, const bool require_alt = false) [static]

Returns true if the given text t contains the entered '&x' shortcut. This method must only be called in handle() methods or callbacks after a keypress event (usually FL_KEYDOWN or FL_SHORTCUT). The given text t (usually a widget's label or menu text) is searched for a '&x' shortcut, and if found, this is compared with the entered key value. Fl::event_text() is used to get the entered key value. Fl::event_state() is used to get the Alt modifier, if require_alt is true.

Parameters

<table>
<thead>
<tr>
<th>t</th>
<th>text or label to search for '&amp;x' shortcut.</th>
</tr>
</thead>
<tbody>
<tr>
<td>require_alt</td>
<td>if true: match only if Alt key is pressed.</td>
</tr>
</tbody>
</table>

Returns

true, if the entered text matches the '&x' shortcut in t, false (0) otherwise.

Note
Useful when a widget's handle(int) method needs dedicated processing of FL_SHORTCUT.

12.160.4.106 tooltip() [1/2]

const char * Fl_Widget::tooltip ( ) const [inline]

Gets the current tooltip text.

Returns

a pointer to the tooltip text or NULL

See also

tooltip(const char*), copy_tooltip(const char*)

12.160.4.107 tooltip() [2/2]

void Fl_Widget::tooltip (const char * text )

Sets the current tooltip text.
Sets a string of text to display in a popup tooltip window when the user hovers the mouse over the widget. The string is *not* copied, so make sure any formatted string is stored in a static, global, or allocated buffer. If you want a copy made and managed for you, use the `copy_tooltip()` method, which will manage the tooltip string automatically. If no tooltip is set, the tooltip of the parent is inherited. Setting a tooltip for a group and setting no tooltip for a child will show the group's tooltip instead. To avoid this behavior, you can set the child's tooltip to an empty string (""").

**Parameters**

| In | text | New tooltip text (no copy is made) |

See also

- `copy_tooltip(const char*)`, `tooltip()`

### 12.160.4.108 top_window()

```cpp
Fl_Widget::top_window ( ) const
```

Returns a pointer to the top-level window for the widget.

In other words, the 'window manager window' that contains this widget. This method differs from `window()` in that it won't return sub-windows (if there are any).

**Returns**

the top-level window, or NULL if no top-level window is associated with this widget.

See also

- `window()`

### 12.160.4.109 top_window_offset()

```cpp
Fl_Widget::top_window_offset ( int & xoff, int & yoff ) const
```

Finds the x/y offset of the current widget relative to the top-level window.

**Parameters**

| Out | xoff, yoff | Returns the x/y offset |

**Returns**

the top-level window (or NULL for a widget that's not in any window)

### 12.160.4.110 type() [1/2]

```cpp
Fl_Widget::type ( ) const [inline]
```

Gets the widget type.

Returns the widget type value, which gives some information about the derived widget class to which the object belongs. Noticeably, the condition `type() >= FL_WINDOW` indicates a widget is an `Fl_Widget` or derived object.

### 12.160.4.111 type() [2/2]

```cpp
Fl_Widget::type ( uchar t ) [inline]
```

Sets the widget type.
12.160.4.112 user_data()

void * Fl_Widget::user_data ( ) const [inline]

Gets the user data for this widget.
Gets the current user data (void *) argument that is passed to the callback function.

Returns
    user data as a pointer

12.160.4.113 visible()

unsigned int Fl_Widget::visible ( ) const [inline]

Returns whether a widget is visible.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the widget is not drawn and hence invisible.</td>
</tr>
</tbody>
</table>

See also
    show(), hide(), visible_r()

12.160.4.114 visible_focus() [1/2]

unsigned int Fl_Widget::visible_focus ( ) const [inline]

Checks whether this widget has a visible focus.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if this widget has no visible focus.</td>
</tr>
</tbody>
</table>

See also
    visible_focus(int), set_visible_focus(), clear_visible_focus()

12.160.4.115 visible_focus() [2/2]

void Fl_Widget::visible_focus ( int v ) [inline]

Modifies keyboard focus navigation.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int v</td>
<td>set or clear visible focus</td>
</tr>
</tbody>
</table>

See also
    set_visible_focus(), clear_visible_focus(), visible_focus()

12.160.4.116 visible_r()

int Fl_Widget::visible_r ( ) const
Returns whether a widget and all its parents are visible.

Return values

0 if the widget or any of its parents are invisible.

See also

show(), hide(), visible()

12.160.4.117 w() [1/2]

int Fl_Widget::w ( ) const [inline]

Gets the widget width.

Returns

the width of the widget in pixels.

12.160.4.118 w() [2/2]

void Fl_Widget::w ( int v ) [inline], [protected]

Internal use only. Use position(int,int), size(int,int) or resize(int,int,int,int) instead.

12.160.4.119 when() [1/2]

Fl_When Fl_Widget::when ( ) const [inline]

Returns the conditions under which the callback is called.
You can set the flags with when(uchar), the default value is FL_WHEN_RELEASE.

Returns

set of flags

See also

when(uchar), Fl_When, do_callback(), Fl::callback_reason()

12.160.4.120 when() [2/2]

void Fl_Widget::when ( uchar i ) [inline]

Sets the flags used to decide when a callback is called.
This controls when callbacks are done. The following values are useful, the default value is FL_WHEN_RELEASE:

- 0: The callback is not done, but changed() is turned on.
- FL_WHEN_CHANGED: The callback is done each time the text is changed by the user.
- FL_WHEN_RELEASE: The callback will be done when this widget loses the focus, including when the window is unmapped. This is a useful value for text fields in a panel where doing the callback on every change is wasteful. However the callback will also happen if the mouse is moved out of the window, which means it should not do anything visible (like pop up an error message). You might do better setting this to zero, and scanning all the items for changed() when the OK button on a panel is pressed.
- FL_WHEN_ENTER_KEY: If the user types the Enter key, the entire text is selected, and the callback is done if the text has changed. Normally the Enter key will navigate to the next field (or insert a newline for a Fl_Multiline_Input) - this changes the behavior.
• **FL_WHEN_ENTER_KEY**/**FL_WHEN_NOT_CHANGED**: The Enter key will do the callback even if the text has not changed. Useful for command fields.

• **FL_WHEN_CLOSED**: If the user requests that the widget is closed, the callback is called with **FL_REASON_CLOSED**. The **Fl_Tabs** widget checks this flag on its children to determine whether to display a close button on the tab of that widget.

**Fl_Widget::when()** is a set of bitflags used by subclasses of **Fl_Widget** to decide when to do the callback. If the value is zero then the callback is never done. Other values are described in the individual widgets. This field is in the base class so that you can scan a panel and **do_callback()** on all the ones that don't do their own callbacks in response to an "OK" button.

**Parameters**

| in | i | set of flags |

See also

**Fl_When, do_callback(), Fl::callback_reason()**

**12.160.4.121 window()**

**Fl_Window * Fl_Widget::window ( ) const**

Returns a pointer to the nearest parent window up the widget hierarchy. This will return sub-windows if there are any, or the parent window if there's no sub-windows. If this widget IS the top-level window, NULL is returned.

**Return values**

| NULL | if no window is associated with this widget |

**Note**

for an **Fl_Window** widget, this returns its parent window (if any), not this window.

See also

**top_window()**

**12.160.4.122 x() [1/2]**

**int Fl_Widget::x ( ) const [inline]**

Gets the widget position in its window.

**Returns**

the x position relative to the window

**12.160.4.123 x() [2/2]**

**void Fl_Widget::x (**

| int v | [inline], [protected] |

Internal use only.

Use **position(int,int)**, **size(int,int)** or **resize(int,int,int,int)** instead.
12.160.4.124 y() [1/2]

int Fl_Widget::y ( ) const [inline]

Gets the widget position in its window.

Returns

the y position relative to the window

12.160.4.125 y() [2/2]

void Fl_Widget::y ( int v ) [inline], [protected]

Internal use only.

Use position(int,int), size(int,int) or resize(int,int,int) instead.

The documentation for this class was generated from the following files:

• Fl_Widget.H
• Fl.cxx
• fl_boxtype.cxx
• fl_labeltype.cxx
• fl_shortcut.cxx
• Fl_Tooltip.cxx
• Fl_Widget.cxx
• Fl_Window.cxx

12.161 Fl_Widget_Surface Class Reference

A surface on which any FLTK widget can be drawn.

#include <Fl_Widget_Surface.H>

Inheritance diagram for Fl_Widget_Surface:

```plaintext
Fl_Surface_Device
Fl_Widget_Surface
 Fl_Copy_Surface
Fl_EPS_File_Surface
Fl_Image_Surface
Fl_Paged_Device
Fl_SVG_File_Surface
Fl_PostScript_File_Device
Fl_Printer
```

Public Member Functions

- void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)
  
  Draws the widget on the drawing surface.

- void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)
  
  Draws a window with its title bar and frame if any.

- virtual void origin (int *x, int *y)
  
  Computes the coordinates of the current origin of graphics functions.

- virtual void origin (int x, int y)
  
  Sets the position of the origin of graphics in the drawable part of the drawing surface.

- void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)
  
  Draws a rectangular part of an on-screen window.

- virtual int printable_rect (int *w, int *h)
  
  Computes the width and height of the drawable area of the drawing surface.

- virtual void translate (int x, int y)
  
  Translates the current graphics origin accounting for the current rotation.

- virtual void untranslate ()
  
  Undoes the effect of a previous translate() call.
Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver ∗ driver ()
  
  Returns the graphics driver of this drawing surface.

- virtual bool is_current ()
  
  Is this surface the current drawing surface?

- virtual void set_current (void)
  
  Make this surface the current drawing surface.

- virtual ∼ Fl_Surface_Device ()
  
  The destructor.

Protected Member Functions

- Fl_Widget_Surface (Fl_Graphics_Driver ∗d)
  
  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- void driver (Fl_Graphics_Driver ∗graphics_driver)
  
  Sets the graphics driver of this drawing surface.

- virtual void end_current ()
  
  FLTK calls this each time a surface ceases to be the current drawing surface.

- Fl_Surface_Device (Fl_Graphics_Driver ∗graphics_driver)
  
  Constructor that sets the graphics driver to use for the created surface.

Protected Attributes

- int x_offset
  
  Horizontal offset to the origin of graphics coordinates

- int y_offset
  
  Vertical offset to the origin of graphics coordinates

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device ∗ pop_current ()
  
  Removes the top element from the current drawing surface stack, and makes the new top element current.

- static void push_current (Fl_Surface_Device ∗new_current)
  
  Pushes new_current on top of the stack of current drawing surfaces, and makes it current.

- static Fl_Surface_Device ∗ surface ()
  
  The current drawing surface.

12.161.1 Detailed Description

A surface on which any FLTK widget can be drawn.

12.161.2 Constructor & Destructor Documentation

12.161.2.1 Fl_Widget_Surface()

Fl_Widget_Surface::Fl_Widget_Surface ( 
                    Fl_Graphics_Driver ∗ d ) [protected]

The constructor.
Parameters

\(d\) can be nul.

12.161.3 Member Function Documentation

12.161.3.1 draw()

```cpp
void Fl_Widget_Surface::draw ( Fl_Widget * widget, int delta_x = 0, int delta_y = 0 )
```

Draws the widget on the drawing surface.
The widget's position on the surface is determined by the last call to `origin()` and by the optional `delta_x` and `delta_y` arguments. Its dimensions are in points unless there was a previous call to `scale()`.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>widget</th>
<th>Any FLTK widget (e.g., standard, custom, window).</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>(delta_x, delta_y)</td>
<td>Optional horizontal and vertical offsets for positioning the widget top left relatively to the current origin of graphics.</td>
</tr>
</tbody>
</table>

12.161.3.2 draw_decorated_window()

```cpp
void Fl_Widget_Surface::draw_decorated_window ( Fl_Window * win, int win_offset_x = 0, int win_offset_y = 0 )
```

Draws a window with its title bar and frame if any. `win_offset_x` and `win_offset_y` are optional coordinates of where to position the window top left. Equivalent to `draw()` if `win` is a subwindow or has no border. Use `Fl_Window::decorated_w()` and `Fl_Window::decorated_h()` to get the size of the framed window.

12.161.3.3 origin() [1/2]

```cpp
void Fl_Widget_Surface::origin ( int * x, int * y ) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

| out | \(x,y\) | If non-null, \(+x\) and \(+y\) are set to the horizontal and vertical coordinates of the graphics origin. |


12.161.3.4 origin() [2/2]

```cpp
void Fl_Widget_Surface::origin ( int x, int y ) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.
Arguments should be expressed relatively to the result of a previous `printable_rect()` call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive `origin()` calls don't combine their effects. Origin() calls are not affected by `rotate()` calls (for classes derived
from Fl_Paged_Device).

Parameters

\begin{center}
\begin{tabular}{|l|}
\hline
in \texttt{x,y} & Horizontal and vertical positions in the drawing surface of the desired origin of graphics. \\
\hline
\end{tabular}
\end{center}


12.161.3.5 print_window_part()

\begin{verbatim}
void Fl_Widget_Surface::print_window_part ( 
   Fl_Window * win, 
   int x, 
   int y, 
   int w, 
   int h, 
   int delta_x = 0, 
   int delta_y = 0 )
\end{verbatim}

Draws a rectangular part of an on-screen window.

Parameters

\begin{center}
\begin{tabular}{|l|l|}
\hline
\texttt{win} & The window from where to capture. Can be an Fl_Gl_Window. Sub-windows that intersect 
   the rectangle are also captured. \\
\texttt{x} & The rectangle left \\
\texttt{y} & The rectangle top \\
\texttt{w} & The rectangle width \\
\texttt{h} & The rectangle height \\
\texttt{delta_x,delta_y} & Optional horizontal and vertical offsets from current graphics origin where to draw the top left 
   of the captured rectangle. \\
\hline
\end{tabular}
\end{center}

12.161.3.6 printable_rect()

\begin{verbatim}
int Fl_Widget_Surface::printable_rect ( 
   int * w, 
   int * h ) [virtual]
\end{verbatim}

Computes the width and height of the drawable area of the drawing surface. Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error


12.161.3.7 translate()

\begin{verbatim}
void Fl_Widget_Surface::translate ( 
   int x, 
   int y ) [virtual]
\end{verbatim}

Translates the current graphics origin accounting for the current rotation. Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.

12.161.3.8 untranslate()

```cpp
void Fl_Widget_Surface::untranslate (
    void ) [virtual]
```

Undoes the effect of a previous translate() call.


The documentation for this class was generated from the following files:

- Fl_Widget_Surface.H
- Fl_Widget_Surface.cxx

12.162 Fl_Widget_Tracker Class Reference

This class should be used to control safe widget deletion.

```cpp
#include <Fl.H>
```

Public Member Functions

- int deleted ()
  
  Returns 1, if the watched widget has been deleted.

- int exists ()
  
  Returns 1, if the watched widget exists (has not been deleted).

- Fl_Widget_Tracker (Fl_Widget *wi)
  
  The constructor adds a widget to the watch list.

- Fl_Widget * widget ()
  
  Returns a pointer to the watched widget.

- ~Fl_Widget_Tracker ()
  
  The destructor removes a widget from the watch list.

12.162.1 Detailed Description

This class should be used to control safe widget deletion.

You can use an Fl_Widget_Tracker object to watch another widget, if you need to know whether this widget has been deleted during a callback.

This simplifies the use of the “safe widget deletion” methods Fl::watch_widget_pointer() and Fl::release_widget_pointer() and makes their use more reliable, because the destructor automatically releases the widget pointer from the widget watch list.

Fl_Widget_Tracker is intended to be used as an automatic (local/stack) variable, such that its destructor is called when the object’s scope is left. This ensures that no stale widget pointers are left in the widget watch list (see example below).

You can also create Fl_Widget_Tracker objects with `new`, but then it is your responsibility to delete the object (and thus remove the widget pointer from the watch list) when it is no longer needed.

**Example:**

```cpp
int MyClass::handle (int event) {
    if (...) {
        Fl_Widget_Tracker wp{this}; // watch myself
        do_callback(); // call the callback
        if (wp.deleted()) return 1; // exit, if deleted
        // Now we are sure that the widget has not been deleted,
        // and it is safe to access the widget:
        box(FL_FLAT_BOX);
        color(Fl_WHITE);
        redraw();
    }
}
```

Generated by Doxygen
12.162.2 Member Function Documentation

12.162.2.1 deleted()

```cpp
int Fl_Widget_Tracker::deleted () [inline]
```
Returns 1, if the watched widget has been deleted.
This is a convenience method. You can also use something like
if (wp.widget() == 0) // ... where `wp` is an `Fl_Widget_Tracker` object.

12.162.2.2 exists()

```cpp
int Fl_Widget_Tracker::exists ( ) [inline]
```
Returns 1, if the watched widget exists (has not been deleted).
This is a convenience method. You can also use something like
if (wp.widget() != 0) // ... where `wp` is an `Fl_Widget_Tracker` object.

12.162.2.3 widget()

```cpp
Fl_Widget * Fl_Widget_Tracker::widget ( ) [inline]
```
Returns a pointer to the watched widget.
The documentation for this class was generated from the following files:

- `FL.H`
- `FL.cxx`

12.163 Fl_Window Class Reference

This widget produces an actual window.
```cpp
#include <Fl_Window.H>
```
Inheritance diagram for Fl_Window:

```
Fl_Window
   |           |
   |           |
Fl_Group  Fl_Widget
   |   +------+
   |         |
Fl_Double_Window Fl_Gl_Window Fl_Single_Window
   |   |   |
   |   |   Fl_Cairo_Window Fl_Overlay_Window Fl_Glut_Window Fl_Menu_Window
```

Public Types

- `typedef struct HICON__ * HICON`

Public Member Functions

- `virtual class Fl_Double_Window * as_double_window ()`  
  Return non-null if this is an Fl_Double_Window object.
- `virtual class Fl_Overlay_Window * as_overlay_window ()`  
  Return non-null if this is an Fl_Overlay_Window object.
- `Fl_Window const * as_window () const FL_OVERRIDE`

Generated by Doxygen
• **Fl_Window * as_window () FL_OVERRIDE**
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• **unsigned int border () const**
  
  Returns whether the window possesses a border.

• **void border (int b)**
  
  Sets whether or not the window manager border is around the window.

• **void clear_border ()**
  
  Fast inline function to turn the window manager border off.

• **void clear_modal_states ()**
  
  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.

• **void copy_label (const char * a)**
  
  Sets the window titlebar label to a copy of a character string.

• **void cursor (const Fl_RGB_Image *, int, int)**
  
  Changes the cursor for this window using the provided image as cursor's shape.

• **void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)**
  
  For back compatibility only.

• **void cursor (Fl_Cursor)**
  
  Changes the cursor for this window.

• **int decorated_h () const**
  
  Returns the window height including any window title bar and any frame added by the window manager.

• **int decorated_w () const**
  
  Returns the window width including any frame added by the window manager.

• **void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)**
  
  For back compatibility only.

• **void default_cursor (Fl_Cursor)**
  
  Sets the default window cursor.

• **void draw_backdrop ()**
  
  Draw the background image if one is set and is aligned inside.

• **Fl_Window (int w, int h, const char * title=0)**
  
  Creates a window from the given width w, height h, and title.

• **Fl_Window (int x, int y, int w, int h, const char * title=0)**
  
  Creates a window from the given position (x, y), size (w, h) and title.

• **void free_position ()**
  
  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.

• **void fullscreen ()**
  
  Makes the window completely fill one or more screens, without any window manager border visible.

• **unsigned int fullscreen_active () const**
  
  Returns non zero if FULLSCREEN flag is set, 0 otherwise.

• **void fullscreen_off ()**
  
  Turns off any side effects of fullscreen()

• **void fullscreen_off (int X, int Y, int W, int H)**
  
  Turns off any side effects of fullscreen() and does resize(x,y,w,h).

• **void fullscreen_screens (int top, int bottom, int left, int right)**
  
  Sets which screens should be used when this window is in full screen mode.

• **int handle (int) FL_OVERRIDE**
  
  Handles the specified event.

• **void hide () FL_OVERRIDE**
  
  Removes the window from the screen.

• **void hotspot (const Fl_Widget & p, int offscreen=0)**
  
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
• void hotspot (const Fl_Widget *, int offscreen=0)
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
• void hotspot (int x, int y, int offscreen=0)
  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.
• const void * icon () const
  Gets the current icon window target dependent data.
• void icon (const Fl_RGB_Image *)
  Sets or resets a single window icon.
• void icon (const void * ic)
  Platform-specific method to set the window icon usable on Windows and X11 only.
• void iconize ()
  Iconifies the window.
• const char * iconlabel () const
  See void Fl_Window::iconlabel(const char *)
• void iconlabel (const char *)
  Sets the icon label.
• void icons (const Fl_RGB_Image *[][], int)
  Sets the window icons.
• void icons (HICON big_icon, HICON small_icon)
  Sets the window icons using HICON handles (Windows platform only).
• const char * label () const
  See void Fl_Window::label(const char *)
• void label (const char *)
  Sets the window title bar label.
• void label (const char *label, const char *iconlabel)
  Sets the icon label.
• void make_current ()
  Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.
• void maximize ()
  Maximizes a top-level window to its current screen.
• unsigned int maximize_active () const
  Returns whether the window is currently maximized.
• unsigned int menu_window () const
  Returns true if this window is a menu window.
• unsigned int modal () const
  Returns true if this window is modal.
• unsigned int non_modal () const
  Returns true if this window is modal or non-modal.
• fl_uintptr_t os_id ()
  Returns a platform-specific identification of a shown window, or 0 if not shown.
• unsigned int override () const
  Returns non zero if OVERRIDE flag is set, 0 otherwise.
• void resize (int X, int Y, int W, int H) FL_OVERRIDE
  Changes the size and position of the window.
• int screen_num ()
  The number of the screen containing the mapped window.
• void screen_num (int screen_num)
  Set the number of the screen where to map the window.
• void **set_menu_window** ()
  Marks the window as a menu window.

• void **set_modal** ()
  A "modal" window, when **shown()**, will prevent any events from being delivered to other windows in the same program,
  and will also remain on top of the other windows (if the X window manager supports the "transient for" property).

• void **set_non_modal** ()
  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a **modal()** one in that it remains on
  top, but it has no effect on event delivery.

• void **set_override** ()
  Activates the flags NOBORDER|OVERRIDE.

• void **set_tooltip_window** ()
  Marks the window as a tooltip window.

• const Fl_Image **shape** ()
  Returns the image controlling the window shape or NULL.

• void **shape** (const Fl_Image &b)
  Set the window’s shape with an Fl_Image.

• void **shape** (const Fl_Image *img)
  Assigns a non-rectangular shape to the window.

• void **show** () FL_OVERRIDE
  Puts the window on the screen.

• void **show** (int argc, char **argv)
  Puts the window on the screen with **show()** and parses command-line arguments.

• int **shown** ()
  Returns non-zero if **show()** has been called (but not **hide()**).

• void **size_range** (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)
  Sets the allowable range the user can resize this window to.

• unsigned int **tooltip_window** () const
  Returns true if this window is a tooltip window.

• void **un_maximize** ()
  Returns a previously maximized top-level window to its previous size.

• void **wait_for_expose** ()
  Waits for the window to be displayed after calling **show()**.

• int **x_root** () const
  Gets the x position of the window on the screen.

• const char * **xclass** () const
  Returns the xclass for this window, or a default.

• void **xclass** (const char *c)
  Sets the xclass for this window.

• int **y_root** () const
  Gets the y position of the window on the screen.

• virtual ~**Fl_Window** ()
  The destructor also deletes all the children.

Public Member Functions inherited from **Fl_Group**

• Fl_Widget *& **_ddfdesign_kludge** ()
  This is for forms compatibility only.

• void **add** (Fl_Widget &)
  The widget is removed from its current group (if any) and then added to the end of this group.

• void **add** (Fl_Widget *)
  See void **Fl_Group::add(Fl_Widget &w)**
• void **add_resizable** (Fl_Widget &o)
  Adds a widget to the group and makes it the resizable widget.

• Fl_Widget **const** + array () **const**
  Returns a pointer to the array of children.

• Fl_Group **const** + as_group () **const** FL_OVERRIDE

• Fl_Group **const** + as_group () FL_OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• void begin ()
  Sets the current group so you can build the widget tree by just constructing the widgets.

• Fl_Widget * child (int n) **const**
  Returns array()[n].

• int children () **const**
  Returns how many child widgets the group has.

• void clear ()
  Deletes all child widgets from memory recursively.

• unsigned int clip_children ()
  Returns the current clipping mode.

• void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

• virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.

• void end ()
  Exactly the same as current(this->parent()).

• int find (const Fl_Widget &o) **const**
  See int Fl_Group::find(const Fl_Widget *w) **const**.

• int find (const Fl_Widget *) **const**
  Searches the child array for the widget and returns the index.

• Fl_Group (int, int, int, int, const char *) =0)
  Creates a new Fl_Group widget using the given position, size, and label string.

• void focus (Fl_Widget *W)

• void forms_end ()
  This is for forms compatibility only.

• void init_sizes ()
  Resets the internal array of widget sizes and positions.

• void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.

• void insert (Fl_Widget &o, Fl_Widget *before)
  This does insert(w, find(before)).

• void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.

• void remove (Fl_Widget *o)
  Removes the widget o from the group.

• void remove (int index)
  Removes the widget at index from the group but does not delete it.

• Fl_Widget * resizable () **const**
  Returns the group's resizable widget.

• void resizable (Fl_Widget &o)
  Sets the group's resizable widget.

• void resizable (Fl_Widget *)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

• virtual ~Fl_Group ()
  The destructor also deletes all the children.
Public Member Functions inherited from `Fl_Widget`

- `void _clear_fullscreen ();`
- `void _set_fullscreen ();`
- `void activate ()` (Activates the widget.)
- `unsigned int active () const` (Returns whether the widget is active.)
- `int active_r () const` (Returns whether the widget and all of its parents are active.)
- `Fl_Align align () const` (Gets the label alignment.)
- `void align (Fl_Align alignment)` (Sets the label alignment.)
- `long argument () const` (Gets the current user data (long) argument that is passed to the callback function.)
- `void argument (long v)` (Sets the current user data (long) argument that is passed to the callback function.)
- `virtual class Fl_Gl_Window * as_gl_window ()` (Returns an `Fl_Gl_Window` pointer if this widget is an `Fl_Gl_Window`.)
- `virtual class Fl_Gl_Window const * as_gl_window () const`
- `void bind_deimage (Fl_Image * img)` (Sets the image to use as part of the widget label when in the inactive state.)
- `void bind_deimage (int f)` (Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.)
- `void bind_image (Fl_Image * img)` (Sets the image to use as part of the widget label when in the active state.)
- `void bind_image (int f)` (Bind the image to the widget, so the widget will delete the image when it is no longer needed.)
- `Fl_Boxtype box () const` (Gets the box type of the widget.)
- `void box (Fl_Boxtype new_box)` (Sets the box type for the widget.)
- `Fl_Callback_p callback () const` (Gets the current callback function for the widget.)
- `void callback (Fl_Callback * cb)` (Sets the current callback function for the widget.)
- `void callback (Fl_Callback * cb, Fl_Callback_User_Data *p, bool auto_free)` (Sets the current callback function and managed user data for the widget.)
- `void callback (Fl_Callback * cb, void *p)` (Sets the current callback function and data for the widget.)
- `void callback (Fl_Callback0 * cb)` (Sets the current callback function for the widget.)
- `void callback (Fl_Callback1 * cb, long p=0)` (Sets the current callback function for the widget.)
- `unsigned int changed () const` (Checks if the widget value changed since the last callback.)
- `void clear_active ()` (Marks the widget as inactive without sending events or changing focus.)
- `void clear_changed ()` (Marks the value of the widget as unchanged.)
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Enables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.
• void copy_label (const char *new_label)
  Sets the current label.
• void copy_tooltip (const char *text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image *deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Imag e *deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.
• int deimage_bound () const
  Returns whether the inactive image is managed by the widget.
• void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with default arguments.
• void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
  Calls the widget callback function with arbitrary arguments.
• void draw_label (int, int, int, Fl_Align) const
Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h () const**
  
  Gets the widget height.

- **Fl_Image * image ()**
  
  Gets the image that is used as part of the widget label when in the active state.

- **const Fl_Image * image () const**
  
  Gets the image that is used as part of the widget label when in the active state.

- **void image (Fl_Image &img)**
  
  Sets the image to use as part of the widget label when in the active state.

- **void image (Fl_Image *img)**
  
  Sets the image to use as part of the widget label when in the active state.

- **int image_bound () const**
  
  Returns whether the image is managed by the widget.

- **int inside (const Fl_Widget *wgt) const**
  
  Checks if this widget is a child of `wgt`.

- **int is_label_copied () const**
  
  Returns whether the current label was assigned with `copy_label()`.

- **const char * label () const**
  
  Gets the current label text.

- **void label (const char *text)**
  
  Sets the current label pointer.

- **void label (Fl_Labeltype a, const char *b)**
  
  Shortcut to set the label text and type in one call.

- **Fl_Color labelcolor () const**
  
  Gets the label color.

- **void labelcolor (Fl_Color c)**
  
  Sets the label color.

- **Fl_Font labelfont () const**
  
  Gets the font to use.

- **void labelfont (Fl_Font f)**
  
  Sets the font to use.

- **Fl_Fontsize labelsize () const**
  
  Gets the font size in pixels.

- **void labelsize (Fl_Fontsize pix)**
  
  Sets the font size in pixels.

- **Fl_Labeltype labeltype () const**
  
  Gets the label type.

- **void labeltype (Fl_Labeltype a)**
  
  Sets the label type.

- **void measure_label (int &ww, int &hh) const**
  
  Sets width `ww` and height `hh` accordingly with the label size.

- **bool needs_keyboard () const**
  
  Returns whether this widget needs a keyboard.

- **void needs_keyboard (bool needs)**
  
  Sets whether this widget needs a keyboard.

- **unsigned int output () const**
  
  Returns if a widget is used for output only.

- **Fl_Group * parent () const**
  
  Returns a pointer to the parent widget.

- **void parent (Fl_Group *p)**
  
  Internal use only - "for hacks only".

Generated by Doxygen
• void **position** (int X, int Y)
  Repositions the window or widget.

• void **redraw** ()
  Schedules the drawing of the widget.

• void **redraw_label** ()
  Schedules the drawing of the label.

• Fl_Color **selection_color** () const
  Gets the selection color.

• void **selection_color** (Fl_Color a)
  Sets the selection color.

• void **set_active** ()
  Marks the widget as active without sending events or changing focus.

• void **set_changed** ()
  Marks the value of the widget as changed.

• void **set_output** ()
  Sets a widget to output only.

• void **set_visible** ()
  Makes the widget visible.

• void **set_visible_focus** ()
  Enables keyboard focus navigation with this widget.

• int **shortcut_label** () const
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.

• void **shortcut_label** (int value)
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.

• void **size** (int W, int H)
  Changes the size of the widget.

• int **take_focus** ()
  Gives the widget the keyboard focus.

• unsigned int **takesevents** () const
  Returns if the widget is able to take events.

• int **test_shortcut** ()
  Returns true if the widget’s label contains the entered ‘&x’ shortcut.

• const char ∗ **tooltip** () const
  Gets the current tooltip text.

• void **tooltip** (const char ∗text)
  Sets the current tooltip text.

• Fl_Window ∗ **top_window** () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window ∗ **top_window_offset** (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar **type** () const
  Gets the widget type.

• void **type** (uchar t)
  Sets the widget type.

• int **use_accents_menu** ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void ∗ **user_data** () const
  Gets the user data for this widget.

• void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)
  Sets the user data for this widget.

• void **user_data** (void ∗v)
Sets the user data for this widget.

- `unsigned int visible () const`
  Returns whether a widget is visible.

- `unsigned int visible_focus () const`
  Checks whether this widget has a visible focus.

- `void visible_focus (int v)`
  Modifies keyboard focus navigation.

- `int visible_r () const`
  Returns whether a widget and all its parents are visible.

- `int w () const`
  Gets the widget width.

- `Fl_When when () const`
  Returns the conditions under which the callback is called.

- `void when (uchar i)`
  Sets the flags used to decide when a callback is called.

- `Fl_Window ∗ window () const`
  Returns a pointer to the nearest parent window up the widget hierarchy.

- `int x () const`
  Gets the widget position in its window.

- `int y () const`
  Gets the widget position in its window.

- `virtual ∼Fl_Widget ()`
  Destroys the widget.

**Static Public Member Functions**

- `static Fl_Window ∗ current ()`
  Returns the last window that was made current.

- `static void default_callback (Fl_Window ∗, void ∗v)`
  Back compatibility: Sets the default callback v for win to call on close event.

- `static void default_icon (const Fl_RGB_Image ∗)`
  Sets a single default window icon.

- `static void default_icons (const Fl_RGB_Image ∗[] , int)`
  Sets the default window icons.

- `static void default_icons (HICON big_icon, HICON small_icon)`
  Sets the default window icons (Windows platform only).

- `static const char ∗ default_xclass ()`
  Returns the default xclass.

- `static void default_xclass (const char ∗)`
  Sets the default window xclass.

- `static bool is_a_rescale ()`
  Returns true when a window is being rescaled.

- `static char show_next_window_iconic ()`
  Returns the static flag whether the next window should be opened iconified.

- `static void show_next_window_iconic (char stat)`
  Sets a static flag whether the next window should be opened iconified.

**Static Public Member Functions inherited from Fl_Group**

- `static Fl_Group ∗ current ()`
  Returns the currently active group.

- `static void current (Fl_Group ∗g)`
  Sets the current group.
Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Member Functions

- void default_size_range ()
  Protected method to calculate the default size range of a window.
- void draw () FL_OVERRIDE
  Draws the widget.
- virtual void flush ()
  Forces the window to be drawn, this window is also made current and calls draw().
- int force_position () const
  Returns the internal state of the window's FORCE_POSITION flag.
- void force_position (int force)
  Sets an internal flag that tells FLTK and the window manager to honor position requests.
- void free_icons ()
  Deletes all icons previously attached to the window.
- int is_resizable ()
  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const
  Forces a child to redraw.
- void draw_children ()
  Draws all children of the group.
- void draw_outside_label (const Fl_Widget &widget) const
  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)
  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)
  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)
  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()
  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const
  Draws a child only if it needs it.
Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
- void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
- void draw_label () const
  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.
- unsigned int flags () const
  Gets the widget flags mask.
- void h (int v)
  Internal use only.
- void set_flag (unsigned int c)
  Sets a flag in the flags mask.
- void w (int v)
  Internal use only.
- void x (int v)
  Internal use only.
- void y (int v)
  Internal use only.

Static Protected Attributes

- static Fl_Window ∗current_
  Stores the last window that was made current.

Friends

- class Fl_Window_Driver
- class Fl_X
Additional Inherited Members

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22,
  AUTO_DELETE_USER_DATA = 1<<23, MAXIMIZED = 1<<24, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

12.163.1 Detailed Description

This widget produces an actual window. This can either be a main window, with a border and title and all the window management controls, or a "subwindow" inside a window. This is controlled by whether or not the window has a parent(). Once you create a window, you usually add children Fl_Widget's to it by using window->add(child) for each new widget. See Fl_Group for more information on how to add and remove children.

There are several subclasses of Fl_Window that provide double-buffering, overlay, menu, and OpenGL support. The window's callback is done if the user tries to close a window using the window manager and Fl::modal() is zero or equal to the window. Fl_Window has a default callback that calls Fl_Window::hide(). Callback reasons can be FL_REASON_CANCELLED if the Escape key was pressed, or FL_REASON_CLOSED when the close button is clicked. FL_WHEN... flags are ignored.

12.163.2 Constructor & Destructor Documentation

12.163.2.1 Fl_Window() [1/2]

Fl_Window::Fl_Window (  
  int w,  
  int h,  
  const char * title = 0 )

Creates a window from the given width w, height h, and title. If Fl_Group::current() is not NULL, the window is created as a subwindow of the parent window. The (w, h) form of the constructor creates a top-level window and asks the window manager to position the window. The (x, y, w, h) form of the constructor either creates a subwindow or a top-level window at the specified location (x, y), subject to window manager configuration. If you do not specify the position of the window, the window manager will pick a place to show the window or allow the user to pick a location. Use position(x, y) or hotspot() before calling show() to request a position on the screen. See Fl_Window::resize() for some more details on positioning windows. Top-level windows initially have visible() set to 0 and parent() set to NULL. Subwindows initially have visible() set to 1 and parent() set to the parent window pointer. Fl_Widget::box() defaults to FL_FLAT_BOX. If you plan to completely fill the window with children widgets you should change this to FL_NO_BOX. If you turn the window border off you may want to change this to FL_UP_BOX.

See also

Fl_Window(int x, int y, int w, int h, const char *title)

12.163.2.2 Fl_Window() [2/2]

Fl_Window::Fl_Window (  
  int x,  
  int y,  
  int w,  
  int h,  
  const char * title = 0 )

Creates a window from the given width w, height h, and title. If Fl_Group::current() is not NULL, the window is created as a subwindow of the parent window. The (w, h) form of the constructor creates a top-level window and asks the window manager to position the window. The (x, y, w, h) form of the constructor either creates a subwindow or a top-level window at the specified location (x, y), subject to window manager configuration. If you do not specify the position of the window, the window manager will pick a place to show the window or allow the user to pick a location. Use position(x, y) or hotspot() before calling show() to request a position on the screen. See Fl_Window::resize() for some more details on positioning windows. Top-level windows initially have visible() set to 0 and parent() set to NULL. Subwindows initially have visible() set to 1 and parent() set to the parent window pointer. Fl_Widget::box() defaults to FL_FLAT_BOX. If you plan to completely fill the window with children widgets you should change this to FL_NO_BOX. If you turn the window border off you may want to change this to FL_UP_BOX.

See also

Fl_Window(int x, int y, int w, int h, const char *title)
int y,
int w,
int h,
const char *title = 0)

Creates a window from the given position (x, y), size (w, h) and title.
On a multi-screen system, the values computed by Fl::screen_xywh(int &X, int &Y, int &W, int &H, int n) can be used
to discover the coordinates of the area of screen #n. When these screens have various scale factor values, an (x, y) pair may not be enough to specify the targeted screen for the window, because the same (x,y) pair can belong to several screens. In that situation, a call to Fl_Window::screen_num(int) is to be used to identify unambiguously the
targeted screen.

See also

Fl_Window(int w, int h, const char ∗title)
Fl::screen_xywh(int &X, int &Y, int &W, int &H, int n)

Note

Under Wayland, it's generally not possible for the client app to control the position of a window in the system.
It's only possible to specify on what screen should the compositor place a fullscreen window.

12.163.2.3 ~Fl_Window()

Fl_Window::~Fl_Window ( ) [virtual]
The destructor also deletes all the children.
This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code.
A kludge has been done so the Fl_Window and all of its children can be automatic (local) variables, but you must
declare the Fl_Window first so that it is destroyed last.

12.163.3 Member Function Documentation

12.163.3.1 as_double_window()

virtual class Fl_Double_Window ∗ Fl_Window::as_double_window ( ) [inline], [virtual]
Return non-null if this is an Fl_Double_Window object.
Reimplemented in Fl_Double_Window.

12.163.3.2 as_overlay_window()

virtual class Fl_Overlay_Window ∗ Fl_Window::as_overlay_window ( ) [inline], [virtual]
Return non-null if this is an Fl_Overlay_Window object.
Reimplemented in Fl_Overlay_Window.

12.163.3.3 as_window() [1/2]

Fl_Window const ∗ Fl_Window::as_window ( ) const [inline], [virtual]
Reimplemented from Fl_Widget.

12.163.3.4 as_window() [2/2]

Fl_Window ∗ Fl_Window::as_window ( ) [inline], [virtual]
Returns an Fl_Window pointer if this widget is an Fl_Window.
Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Window. If
it returns non-NULL, then the widget in question is derived from Fl_Window, and you can use the returned pointer
to access its children or other Fl_Window-specific methods.

Return values

NULL if this widget is not derived from Fl_Window.
Note

This method is provided to avoid dynamic_cast.

See also

Fl_Widget::as_group(), Fl_Widget::as_gl_window()

Reimplemented from Fl_Widget.

12.163.3.5 border()

```cpp
void Fl_Window::border (int b)
```

Sets whether or not the window manager border is around the window.
The default value is true. With some X window managers, this does not work after show() has been called.

12.163.3.6 clear_border()

```cpp
void Fl_Window::clear_border () [inline]
```

Fast inline function to turn the window manager border off.
It only works before show() is called.

12.163.3.7 clear_modal_states()

```cpp
void Fl_Window::clear_modal_states () [inline]
```

Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.
Note that there are three states for a window: modal, non-modal, and normal.
You can not change the "modality" of a window whilst it is shown, so it is necessary to first hide() the window, change its "modality" as required, then re-show the window for the new state to take effect.
This method can also be used to change a "modal" window into a "non-modal" one. On several supported platforms, the "modal" state over-rides the "non-modal" state, so the "modal" state must be cleared before the window can be set into the "non-modal" state. In general, the following sequence should work:

```cpp
win->hide();
win->clear_modal_states();
// Set win to new state as desired, or leave "normal", e.g...
win->set_non_modal();
win->show();
```

Note

Under some window managers, the sequence of hiding the window and changing its modality will often cause it to be re-displayed at a different position when it is subsequently shown. This is an irritating feature but appears to be unavoidable at present. As a result we would advise to use this method only when absolutely necessary.

See also

void set_modal(), void set_non_modal()

12.163.3.8 current()

```cpp
FL_Window ∗ Fl_Window::current () [static]
```

Returns the last window that was made current.

See also

Fl_Window::make_current()
12.163.3.9 cursor() [1/3]

```cpp
void Fl_Window::cursor {
    const Fl_RGB_Image * image,
    int hotx,
    int hoty }
```

Changes the cursor for this window using the provided image as cursor's shape.
The window must be show()n for this function to have any effect. This always calls the system. If you are changing
the cursor a lot you may want to keep track of how you set it in a static variable and call this only if the new cursor
is different.
The default cursor will be used if the provided image cannot be used as a cursor.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>Sets the cursor size and shape</td>
</tr>
<tr>
<td>hotx,hoty</td>
<td>Sets the cursor's active location relatively to top-left of image when clicking</td>
</tr>
</tbody>
</table>

See also

cursor(Fl_Cursor), default_cursor()

12.163.3.10 cursor() [2/3]

```cpp
void Fl_Window::cursor {
    Fl_Cursor c,
    Fl_Color ,
    Fl_Color = FL_WHITE }
```

For back compatibility only.
Same as Fl_Window::cursor(Fl_Cursor)

12.163.3.11 cursor() [3/3]

```cpp
void Fl_Window::cursor {
    Fl_Cursor c }
```

Changes the cursor for this window.
The window must be show()n for this function to have any effect. This always calls the system. If you are changing
the cursor a lot you may want to keep track of how you set it in a static variable and call this only if the new cursor
is different.
The type Fl_Cursor is an enumeration defined in `<FL/Enumerations.H>`.

See also

cursor(const Fl_RGB_Image*, int, int), default_cursor()

12.163.3.12 decorated_h()

```cpp
int Fl_Window::decorated_h ( ) const
```

Returns the window height including any window title bar and any frame added by the window manager.
Same as h() if applied to a subwindow, or if window is not yet mapped.

Note

Under X11, FLTK is able to compute the size of window titlebars and borders only if these decoration elements are strictly X11-based. When that's not the case, decorated_h() returns the same value as h() and decorated_w() as w(), and FLTK cannot access window decorations.

Under X11 again, the values returned by decorated_h() and decorated_w() may not be reliable during a resize operation. The size of decoration elements of a window is best computed when the window is first mapped.
12.163.3.13  decorated_w()

```cpp
int Fl_Window::decorated_w ( ) const
```

Returns the window width including any frame added by the window manager.  
Same as w() if applied to a subwindow, or if window is not yet mapped.

See also

```cpp
decorated_h().
```

12.163.3.14  default_cursor() [1/2]

```cpp
void Fl_Window::default_cursor ( 
    Fl_Cursor c,
    Fl_Color ,
    Fl_Color = FL_WHITE )
```

For back compatibility only.

same as Fl_Window::default_cursor(Fl_Cursor)

12.163.3.15  default_cursor() [2/2]

```cpp
void Fl_Window::default_cursor ( 
    Fl_Cursor c )
```

Sets the default window cursor.  
This is the cursor that will be used after the mouse pointer leaves a widget with a custom cursor set.

See also

```cpp
cursor(const Fl_RGB_Image*, int, int), default_cursor()
```

12.163.3.16  default_icon()

```cpp
void Fl_Window::default_icon ( 
    const Fl_RGB_Image * icon ) [static]
```

Sets a single default window icon.  
If icon is NULL the current default icons are removed.

**Parameters**

| in | icon | default icon for all windows subsequently created or NULL |

See also

```cpp
Fl_Window::default_icons(const Fl_RGB_Image [][], int)
Fl_Window::icon(const Fl_RGB_Image *)
Fl_Window::icons(const Fl_RGB_Image [], int)
```

**Note**

See Window icons for the Wayland platform.

12.163.3.17  default_icons() [1/2]

```cpp
void Fl_Window::default_icons ( 
    const Fl_RGB_Image * icons[],
    int count ) [static]
```

Sets the default window icons.
The default icons are used for all windows that don't have their own icons set before `show()` is called. You can change the default icons whenever you want, but this only affects windows that are created (and shown) after this call.

The given images in `icons` are copied. You can use a local variable or free the images immediately after this call.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>icons</code></th>
<th>default icons for all windows subsequently created</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>count</code></td>
<td>number of images in <code>icons</code>. Set to 0 to remove the current default icons</td>
</tr>
</tbody>
</table>

See also

- `Fl_Window::default_icon(const Fl_RGB_Image *)`
- `Fl_Window::icon(const Fl_RGB_Image *)`
- `Fl_Window::icons(const Fl_RGB_Image *[], int)`

Note

See Window icons for the Wayland platform.

### 12.163.3.18 default_icons()

```
static void Fl_Window::default_icons (             
    HICON big_icon,                               
    HICON small_icon ) [static]                 
```

Sets the default window icons (Windows platform only).
Convenience function to set the default icons using Windows' native HICON icon handles.
The given icons are copied. You can free the icons immediately after this call.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>big_icon</code></th>
<th>default large icon for all windows subsequently created</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>small_icon</code></td>
<td>default small icon for all windows subsequently created</td>
</tr>
</tbody>
</table>

See also

- `Fl_Window::default_icon(const Fl_RGB_Image *)`
- `Fl_Window::default_icons(const Fl_RGB_Image *[], int)`
- `Fl_Window::icon(const Fl_RGB_Image *)`
- `Fl_Window::icons(const Fl_RGB_Image *[], int)`
- `Fl_Window::icons(HICON, HICON)`

### 12.163.3.19 default_size_range()

```
void Fl_Window::default_size_range ( ) [protected]
```

Protected method to calculate the default size range of a window.
This method is called internally prior to showing a window to ensure that the window's size range values are calculated if a `resizable()` widget has been set but `size_range()` has not been called explicitly.
This method does nothing if `size_range()` has been called before.
Otherwise FLTK tries to figure out the window's size range from the setting of the window's `resizable()` widget as follows and roughly in the given order.

1. If `resizable()` is NULL (this is the default) then the window cannot be resized and the resize border and max-size control will not be displayed for the window.
2. If either dimension of `resizable()` is zero, then the window cannot resize in that direction.

3. The `resizable()` widget is clipped to the window area.

4. The non-resizable portion of the window is calculated as the difference of the window's size and the clipped `resizable()` widget's size.

5. If either dimension of the clipped `resizable()` widget is greater than 100, then 100 is considered its minimum width/height. This allows the resizable widget to shrink below its original size.

6. Finally the minimum width/height of the window is set to the non-resizable portion plus the width/height of the `resizable()` widget as calculated above.

In simple words:

- It is assumed that the `resizable()` widget can be indefinitely enlarged and/or shrunk to a minimum width/height of 100 unless it is smaller than that, which is then considered the minimum.

- The window's `size_range()` minimum values are set to the sum of the non-resizable portion of the window and the previously calculated minimum size of the `resizable()` widget.

Examples:

```cpp
Fl_Window win(400, 400);
win.resizable(win);
// win.size_range(100, 100, 0, 0);
The minimum size of the resizable is 100, hence the minimum size of the total window is also 100 in both directions.
Fl_Window win(400, 400);
Fl_Box box(20, 20, 360, 360);
win.resizable(box);
// win.size_range(140, 140, 0, 0);
The calculated minimum width and height would be 20 + 100 + 20 in both dimensions.
Fl_Window win(400, 400);
Fl_Box box(200, 0, 500, 300); // note: width 500 too large: clipped
win.resizable(box);
// win.size_range(300, 200, 0, 0);
The width of the resizable is clipped to 200, hence the minimum size of the total window is also 200 (fix) + 100 (min. resizable) in x direction. The minimum value in y direction is 100 (resizable) + 100 (fixed part).
The calculation is based on clipping the resizable widget to the window area to prevent programming errors and the assumption that the resizable widget can be shrunk to 100x100 or its original size, whichever is smaller.
If this is not what you want, please use `Fl_Window::size_range()` explicitly so you can set any appropriate range.
```

12.163.3.20 default_xclass() [1/2]

```cpp
class Fl_Window{

public:

  const char * default_xclass () [static]

  Returns the default xclass.

  See also

  `Fl_Window::default_xclass(const char *)`

}
```

12.163.3.21 default_xclass() [2/2]

```cpp
void Fl_Window::default_xclass (const char * xc) [static]

Sets the default window xclass.
The default xclass is used for all windows that don't have their own xclass set before `show()` is called. You can change the default xclass whenever you want, but this only affects windows that are created (and shown) after this call.
The given string `xc` is copied. You can use a local variable or free the string immediately after this call.
If you don't call this, the default xclass for all windows will be "FLTK". You can reset the default xclass by specifying `NULL` for `xc`.
If you call `Fl_Window::xclass(const char *)` for any window, then this also sets the default xclass, unless it has been set before.
```
Parameters

| in  | xc | default xclass for all windows subsequently created |

See also

Fl_Window::xclass(const char *)

### 12.163.3.22 draw()

```cpp
void Fl_Window::draw() [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from `Fl_Group`.

### 12.163.3.23 flush()

```cpp
void Fl_Window::flush() [protected], [virtual]
```

Forces the window to be drawn, this window is also made current and calls `draw()`.

Reimplemented in `Fl_Double_Window`, `Fl_Gl_Window`, and `Fl_Overlay_Window`.

### 12.163.3.24 force_position() [1/2]

```cpp
int Fl_Window::force_position() const [inline], [protected]
```

Returns the internal state of the window's FORCE_POSITION flag.

**Return values**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if flag is set</td>
</tr>
<tr>
<td>0</td>
<td>otherwise</td>
</tr>
</tbody>
</table>

See also

force_position(int)

### 12.163.3.25 force_position() [2/2]

```cpp
void Fl_Window::force_position(int force) [inline], [protected]
```

Sets an internal flag that tells FLTK and the window manager to honor position requests.

This is used internally and should not be needed by user code.

**Parameters**

| in  | force | 1 to set the FORCE_POSITION flag, 0 to clear it |

### 12.163.3.26 free_icons()

```cpp
void Fl_Window::free_icons() [protected]
```

Deletes all icons previously attached to the window.

Generated by Doxygen
12.163.3.27 free_position()

void Fl_Window::free_position() [inline]
Undo the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.
This is for Forms compatibility only.

Deprecated please use force_position(0) instead

12.163.3.28 fullscreen()

void Fl_Window::fullscreen()
Makes the window completely fill one or more screens, without any window manager border visible.
You must use fullscreen_off() to undo this.

Note
On some platforms, this can result in the keyboard being grabbed. The window may also be recreated, meaning hide() and show() will be called.

See also
void Fl_Window::fullscreen_screens()

12.163.3.29 fullscreen_screens()

void Fl_Window::fullscreen_screens {
  int top,
  int bottom,
  int left,
  int right
}
Sets which screens should be used when this window is in fullscreen mode.
The window will be resized to the top of the screen with index top, the bottom of the screen with index bottom, etc.
If this method is never called, or if any argument is < 0, then the window will be resized to fill the screen it is currently on.

See also
void Fl_Window::fullscreen()

12.163.3.30 handle()

int Fl_Window::handle() [virtual]
Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to override the behavior of the base class.
This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to filter event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you consumed the event by returning 1 even if you didn't do anything with it.
Parameters

| in  | event | the kind of event received |

Return values

| 0   | if the event was not used or understood |
| 1   | if the event was used and can be deleted |

See also

[Fl_Event](#)

Reimplemented from [Fl_Group](#).

### 12.163.3.31 hide()

```cpp
void Fl_Window::hide ( ) [virtual]
```

Removes the window from the screen.

If the window is already hidden or has not been shown then this does nothing and is harmless.

Reimplemented from [Fl_Widget](#).

### 12.163.3.32 hotspot()

```cpp
void Fl_Window::hotspot ( int x, int y, int offscreen = 0 )
```

Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.

If the optional offscreen parameter is non-zero, then the window is allowed to extend off the screen (this does not work with some X window managers).

See also

[position()](#)

### 12.163.3.33 icon() [1/3]

```cpp
const void * Fl_Window::icon ( ) const
```

Gets the current icon window target dependent data.

**Deprecated** in 1.3.3

### 12.163.3.34 icon() [2/3]

```cpp
void Fl_Window::icon ( const Fl_RGB_Image * icon )
```

Sets or resets a single window icon.

A window icon can be changed while the window is shown, but this may be platform and/or window manager dependent. To be sure that the window displays the correct window icon you should always set the icon before the window is shown.

If a window icon has not been set for a particular window, then the default window icon (see links below) or the system default icon will be used.

This method makes an internal copy of the `icon` pixel buffer, so once set, the `Fl_RGB_Image` instance can be freed by the caller.
Parameters

| in | icon | icon for this window, NULL to reset window icon. |

See also

- `Fl_Window::default_icon(const Fl_RGB_Image *)`
- `Fl_Window::default_icons(const Fl_RGB_Image *[ ], int)`
- `Fl_Window::icons(const Fl_RGB_Image *[ ], int)`

Note

See Window icons for the Wayland platform.

### 12.163.3.35 icon()

```cpp
void Fl_Window::icon (const void * ic )
```

Platform-specific method to set the window icon usable on Windows and X11 only.

See Setting the Icon of a Window for its use under X11, and Setting the Icon of a Window under Windows.

**Deprecated** in 1.3.3 in favor of platform-independent methods `Fl_Window::icon(const Fl_RGB_Image *icon)` and `Fl_Window::icons(const Fl_RGB_Image *icons[ ], int count)`.

### 12.163.3.36 iconize()

```cpp
void Fl_Window::iconize ( )
```

Iconifies the window.

If you call this when `shown()` is false it will `show()` it as an icon. If the window is already iconified this does nothing. Call `show()` to restore the window.

When a window is iconified/restored (either by these calls or by the user) the `handle()` method is called with FL_HIDE and FL_SHOW events and `visible()` is turned on and off.

There is no way to control what is drawn in the icon except with the string passed to `Fl_Window::xclass()`. You should not rely on window managers displaying the icons.

### 12.163.3.37 icons()

```cpp
void Fl_Window::icons (const Fl_RGB_Image * icons[ ], int count )
```

Sets the window icons.

You may set multiple window icons with different sizes. Dependent on the platform and system settings the best (or the first) icon will be chosen.

The given images in `icons` are copied. You can use a local variable or free the images immediately after this call. If `count` is zero, current icons are removed. If `count` is greater than zero (must not be negative), then `icons[]` must contain at least `count` valid image pointers (not NULL). Otherwise the behavior is undefined.

Parameters

| in | icons | icons for this window |
| in | count | number of images in `icons`. Set to 0 to remove the current icons |

See also

- `Fl_Window::default_icon(const Fl_RGB_Image *)`
- `Fl_Window::default_icons(const Fl_RGB_Image *[ ], int)`
Fl_Window::icon(const Fl_RGB_Image *)

Note
See Window icons for the Wayland platform.

12.163.3.38 icons() [2/2]
void Fl_Window::icons (  
    HICON big_icon,  
    HICON small_icon  
)
Sets the window icons using HICON handles (Windows platform only). The given icons are copied. You can free the icons immediately after this call.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>big_icon</th>
<th>large window icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>small_icon</td>
<td>small window icon</td>
</tr>
</tbody>
</table>

12.163.3.39 is_resizable()
int Fl_Window::is_resizable ( ) [protected]
Protected method to determine whether a window is resizable. If size_range() has not yet been called this method calculates the default size range values by calling default_size_range(). This method is for internal use only. The returned value is a bit mask and non-zero if the window is resizable in at least one direction.

Returns
non-zero if the window is resizable

Return values

| 0 | the window is not resizable |
| 1 | the window is resizable in horizontal direction (w) |
| 2 | the window is resizable in vertical direction (h) |
| 3 | the window is resizable in both directions (w and h) |

See also
default_size_range()  

12.163.3.40 make_current()
void Fl_Window::make_current ( )
Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window. This is useful for incremental update of windows, such as in an idle callback, which will make your program behave much better if it draws a slow graphic. Danger: incremental update is very hard to debug and maintain! This method only works for the Fl_Window and Fl_Gl_Window derived classes.

12.163.3.41 maximize()
void Fl_Window::maximize ( )
Maximizes a top-level window to its current screen.
This function is effective only with a show()'n, resizable, top-level window. Bordered and borderless windows can be used.

See also

   Fl_Window::un_maximize(), Fl_Window::maximize_active()

12.163.3.42  modal()

unsigned int Fl_Window::modal ( ) const [inline]
Returns true if this window is modal.

12.163.3.43  os_id()

fl_uintptr_t Fl_Window::os_id ( )
Returns a platform-specific identification of a shown window, or 0 if not shown.

Note

This identification may differ from the platform-specific reference of an Fl_Window object used by functions
fl_x11_xid(), fl_mac_xid(), fl_x11_find(), and fl_mac_find().

   • X11 platform: the window’s XID.
   • macOS platform: The window number of the window's window device.
   • other platforms: 0.

12.163.3.44  resize()

void Fl_Window::resize ( int X, int Y, int W, int H ) [virtual]
Changes the size and position of the window.
If shown() is true, these changes are communicated to the window server (which may refuse that size and cause a further resize). If shown() is false, the size and position are used when show() is called. See Fl_Group for the effect of resizing on the child widgets.
You can also call the Fl_Widget methods size(x,y) and position(w,h), which are inline wrappers for this virtual function.
A top-level window can not force, but merely suggest a position and size to the operating system. The window manager may not be willing or able to display a window at the desired position or with the given dimensions. It is up to the application developer to verify window parameters after the resize request.
Reimplemented from Fl_Group.

12.163.3.45  screen_num()

void Fl_Window::screen_num ( int screen_num )
Set the number of the screen where to map the window.
Call this and set also the window's desired position before show()'ing the window. This can be necessary when a system has several screens with distinct scaling factor values because the window's x() and y() may not suffice to uniquely identify one screen. To see that, consider a system with two screens where the screen at left is A pixel-wide and has a scale factor of 1 whereas the screen at right has a scale factor of 2. For the sake of simplicity, consider only the X coordinates of windows. FLTK coordinates translate directly to pixel coordinates on the left screen, whereas FLTK coordinates multiplied by 2 correspond to pixel coordinates on the right screen. Consequently, FLTK coordinates between A/2 + 1 and A-1 can map to both screens. Both window coordinates and screen number are necessary to uniquely identify where a window is to be mapped.
12.163.3.46 set_menu_window()

void Fl_Window::set_menu_window ( ) [inline]
Marks the window as a menu window.
This is intended for internal use, but it can also be used if you write your own menu handling. However, this is not
recommended.
This flag is used for correct "parenting" of windows in communication with the windowing system. Modern X window
managers can use different flags to distinguish menu and tooltip windows from normal windows.
This must be called before the window is shown and cannot be changed later.

12.163.3.47 set_modal()

void Fl_Window::set_modal ( ) [inline]
A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same
program, and will also remain on top of the other windows (if the X window manager supports the "transient for"
property).
Several modal windows may be shown at once, in which case only the last one shown gets events. You can see
which window (if any) is modal by calling Fl::modal().

12.163.3.48 set_non_modal()

void Fl_Window::set_non_modal ( ) [inline]
A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on
top, but it has no effect on event delivery.
There are three states for a window: modal, non-modal, and normal.

12.163.3.49 set_tooltip_window()

void Fl_Window::set_tooltip_window ( ) [inline]
Marks the window as a tooltip window.
This is intended for internal use, but it can also be used if you write your own tooltip handling. However, this is not
recommended.
This flag is used for correct "parenting" of windows in communication with the windowing system. Modern X window
managers can use different flags to distinguish menu and tooltip windows from normal windows.
This must be called before the window is shown and cannot be changed later.

Note
Since Fl_Tooltip_Window is derived from Fl_Menu_Window, this also clears the menu_window() state.

12.163.3.50 shape() [1/2]

void Fl_Window::shape (const Fl_Image & img )
Set the window's shape with an Fl_Image.
See also

void shape(const Fl_Image* img)

12.163.3.51 shape() [2/2]

void Fl_Window::shape (const Fl_Image * img )
Assigns a non-rectangular shape to the window.
This function gives an arbitrary shape (not just a rectangular region) to an Fl_Window. An Fl_Image of any dimen-
sion can be used as mask; it is rescaled to the window's dimension as needed.
The layout and widgets inside are unaware of the mask shape, and most will act as though the window's rectangular
bounding box is available to them. It is up to you to make sure they adhere to the bounds of their masking shape.
The img argument can be an Fl_Bitmap, Fl_Pixmap, Fl_RGB_Image or Fl_Shared_Image:
With Fl_Bitmap or Fl_Pixmap, the shaped window covers the image part where bitmap bits equal one, or where the pixmap is not fully transparent.

With an Fl_RGB_Image with an alpha channel (depths 2 or 4), the shaped window covers the image part that is not fully transparent.

With an Fl_RGB_Image of depth 1 (gray-scale) or 3 (RGB), the shaped window covers the non-black image part.

With an Fl_Shared_Image, the shape is determined by rules above applied to the underlying image. The shared image should not have been scaled through Fl_Image::scale().

Platform details:

• On the unix/linux platform, the SHAPE extension of the X server is required. This function does control the shape of Fl_Gl_Window instances.

• On the Windows platform, this function does nothing with class Fl_Gl_Window.

• On the Mac platform, OS version 10.4 or above is required. An 8-bit shape-mask is used when img is an Fl_RGB_Image: with depths 2 or 4, the image alpha channel becomes the shape mask such that areas with alpha = 0 are out of the shaped window; with depths 1 or 3, white and black are in and out of the shaped window, respectively, and other colors give intermediate masking scores. This function does nothing with class Fl_Gl_Window.

The window borders and caption created by the window system are turned off by default. They can be re-enabled by calling Fl_Window::border(1).

A usage example is found at example/shapedwindow.cxx.

Version 1.3.3

12.163.3.52 show() [1/2]

void Fl_Window::show ( ) [virtual]

Puts the window on the screen.
This has the side effect of opening the display, if not done before. If the window is already shown then it is restored and raised to the top. This is really convenient because your program can call show() at any time, even if the window is already up. It also means that show() serves the purpose of raise() in other toolkits.

Fl_Window::show(int argc, char **argv) is used for top-level windows and allows standard arguments to be parsed from the command-line.

Note

For some obscure reasons Fl_Window::show() resets the current group by calling Fl_Group::current(0). The comments in the code say "get rid of very common user bug: forgot end()". Although this is true it may have unwanted side effects if you show() an unrelated window (maybe for an error message or warning) while building a window or any other group widget.

Todo Check if we can remove resetting the current group in a later FLTK version (after 1.3.x). This may break "already broken" programs though if they rely on this "feature".

See also

Fl_Window::show(int argc, char **argv)

Reimplemented from Fl_Widget.
void Fl_Window::show (
    int argc,
    char ** argv )
Puts the window on the screen with show() and parses command-line arguments.
This call should be used for top-level windows, at least for the first (main) window. It allows standard arguments to be parsed, as done by Fl::args(int, char **), from the command-line. You can use argc and argv from main(int argc, char ** argv) for this call.
This call also sets up some system-specific internal variables, that is, it sets FL_SELECTION_COLOR and calls Fl::background(), Fl::background2(), Fl::foreground() with default or X resources-given values, and calls Fl::scheme(const char *) for the current scheme. On X11, it also calls Fl::dnd_text_ops(int), Fl::ToolTip::enable(int), Fl::visible_focus(int) with X resources-given values.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argc</td>
<td>command-line argument count, usually from main()</td>
</tr>
<tr>
<td>argv</td>
<td>command-line argument vector, usually from main()</td>
</tr>
</tbody>
</table>

See also

virtual void Fl_Window::show()
    Fl::args(int, char **)

static char Fl_Window::show_next_window_iconic ( ) [inline], [static]
Returns the static flag whether the next window should be opened iconified.

Note

This is an internal function, you should not use this in user code.
Please use Fl_Window::iconize() to iconify a window.

static void Fl_Window::show_next_window_iconic ( char stat ) [inline], [static]
Sets a static flag whether the next window should be opened iconified.

Note

This is an internal function, you should not use this in user code.
Please use Fl_Window::iconize() instead.

int Fl_Window::shown ( ) [inline]
Returns non-zero if show() has been called (but not hide()).
You can tell if a window is iconified with (w->shown() && lw->visible()).

void Fl_Window::size_range ( 
    int minWidth,
    int minHeight,
    int maxWidth = 0,
    int maxHeight = 0,
Sets the allowable range the user can resize this window to.

We recommend to call `size_range()` if you have a `resizable()` widget in a main window, and to call it after setting the `resizable()` and before `show()`ing the window for best cross platform compatibility.

If this function is not called, FLTK tries to figure out the range when the window is shown. Please see the protected method `default_size_range()` for details.

It is undefined what happens if the current window size does not fit in the constraints passed to `size_range()`.

**Note**

This only works for top-level windows and the exact behavior can be platform specific. To work correctly across all platforms `size_range()` must be called after setting the `resizable()` widget of the window and before the window is `show()`'n.

Calling `size_range()` after the window has been shown may work on some but not all platforms. If you need to change the `size_range()` after the window has been shown, then you should consider to `hide()` and `show()` the window again, i.e. call `hide()`, `size_range()`, and `show()` in this order.

Typical usage: call `size_range(minWidth, minHeight);`

after setting the resizable widget and before calling `show()`. This ensures that the window cannot be resized smaller than the given values by user interaction.

`maxWidth` and `maxHeight` might be useful in some special cases but less frequently used.

The other optional parameters `deltaX`, `deltaY`, and `aspectRatio` are not recommended because they may not work on all platforms and may even under X11 not be supported by all Window Managers.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>minWidth,minHeight</code></td>
<td>The smallest the window can be. Either value must be greater than 0.</td>
</tr>
<tr>
<td><code>maxWidth,maxHeight</code></td>
<td>The largest the window can be. If either is equal to the minimum then you cannot resize in that direction. If either is zero then FLTK picks a maximum size in that direction such that the window will fill the screen.</td>
</tr>
<tr>
<td><code>deltaX,deltaY</code></td>
<td>These are size increments. The window will be constrained to widths of <code>minWidth + N * deltaX</code>, where N is any non-negative integer. If these are less or equal to 1 they are ignored (this is always ignored on Windows).</td>
</tr>
<tr>
<td><code>aspectRatio</code></td>
<td>A flag that indicates that the window should preserve its aspect ratio. This only works if both the maximum and minimum have the same aspect ratio (ignored on Windows and by many X window managers).</td>
</tr>
</tbody>
</table>

### 12.163.3.58 un_maximize()

`void Fl_Window::un_maximize ( )`

Returns a previously maximized top-level window to its previous size.

See also

`Fl_Window::maximize()`

### 12.163.3.59 wait_for_expose()

`void Fl_Window::wait_for_expose ( )`

 Waits for the window to be displayed after calling `show()`.

`Fl_Window::show()` is not guaranteed to show and draw the window on all platforms immediately. Instead this is done in the background; particularly on X11 it will take a few messages (client server roundtrips) to display the window. Usually this small delay doesn't matter, but in some cases you may want to have the window instantiated and displayed synchronously.

Currently (as of FLTK 1.3.4) this method has an effect on X11 and Mac OS. On Windows, `show()` is always synchronous. The effect of `show()` varies with versions of Mac OS X: early versions have the window appear on the
screen when `show()` returns, later versions don't. If you want to write portable code and need this synchronous `show()` feature, add `win->wait_for_expose()` on all platforms, and FLTK will just do the right thing.

This method can be used for displaying splash screens before calling `Fl::run()` or for having exact control over which window has the focus after calling `show()`.

If the window is not `shown()`, this method does nothing.

Note

Depending on the platform and window manager `wait_for_expose()` may not guarantee that the window is fully drawn when it is called. Under X11 it may only make sure that the window is mapped, i.e. the internal (OS dependent) window object was created (and maybe shown on the desktop as an empty frame or something like that). You may need to call `Fl::flush()` after `wait_for_expose()` to make sure the window and all its widgets are drawn and thus visible.

FLTK does the best it can do to make sure that all widgets get drawn if you call `wait_for_expose()` and `Fl::flush()`. However, dependent on the window manager it can not be guaranteed that this does always happen synchronously. The only guaranteed behavior that all widgets are eventually drawn is if the FLTK event loop is run continuously, for instance with `Fl::run()`.

See also

```c
virtual void Fl_Window::show()
```

Example code for displaying a window before calling `Fl::run()`

```c
Fl_Double_Window win = new Fl_Double_Window(...);
// do more window initialization here ...
win->show(); // show window
win->wait_for_expose(); // wait, until displayed
Fl::flush(); // make sure everything gets drawn
// do more initialization work that needs some time here ...
Fl::run(); // start FLTK event loop
```

Note that the window will not be responsive until the event loop is started with `Fl::run()`.

### 12.163.3.60 xclass() [1/2]

```c
const char * Fl_Window::xclass() const
```

Returns the xclass for this window, or a default.

See also

```c
Fl_Window::default_xclass(const char *)
Fl_Window::xclass(const char *)
```

### 12.163.3.61 xclass() [2/2]

```c
void Fl_Window::xclass(
    const char * xc
)
```

Sets the xclass for this window. A string used to tell the system what type of window this is. Mostly this identifies the picture to draw in the icon. This only works if called before calling `show()`.

*Under X*, this is turned into a `XA_WM_CLASS` pair by truncating at the first non-alphanumeric character and capitalizing the first character, and the second one if the first is 'x'. Thus "foo" turns into "foo, Foo", and "xprog.1" turns into "xprog, XProg".

*Under Microsoft Windows*, this string is used as the name of the `WNDCLASS` structure, though it is not clear if this can have any visible effect.

Since

FLTK 1.3 the passed string is copied. You can use a local variable or free the string immediately after this call. Note that FLTK 1.1 stores the `pointer` without copying the string.

If the default xclass has not yet been set, this also sets the default xclass for all windows created subsequently.
12.163.4 Member Data Documentation

12.163.4.1 current_

Fl_Window * Fl_Window::current_ [static], [protected]
Stores the last window that was made current.
See current() const
The documentation for this class was generated from the following files:

- Fl_Window.H
- Fl_arg.cxx
- fl_cursor.cxx
- Fl_Window.cxx
- Fl_Window_fullscreen.cxx
- Fl_Window_hotspot.cxx
- Fl_Window_iconize.cxx

12.164 Fl_Wizard Class Reference

This widget is based off the Fl_Tabs widget, but instead of displaying tabs it only changes "tabs" under program control.

#include <Fl_Wizard.H>

Inheritance diagram for Fl_Wizard:

```
Fl_Widget
   |      ___________
   |      |          |
   |      |          |
   v      Fl_Group
         |          |
         v          |
         Fl_Wizard
```

Public Member Functions

- **Fl_Wizard** (int, int, int, int, const char * = 0)
  
  The constructor creates the Fl_Wizard widget at the specified position and size.

- **void next ()**
  
  This method shows the next child of the wizard.

- **void prev ()**
  
  Shows the previous child.

- **Fl_Widget * value ()**
  
  Gets the current visible child widget.

- **void value (Fl_Widget *)**
  
  Sets the child widget that is visible.

Public Member Functions inherited from Fl_Group

- **Fl_Widget * & _ddfdesign_kludge ()**
  
  This is for forms compatibility only.

- **void add (Fl_Widget &)**
  
  The widget is removed from its current group (if any) and then added to the end of this group.

- **void add (Fl_Widget *o)**
See void Fl_Group::add(Fl_Widget &w)
• void add_resizable (Fl_Widget &o)
  Adds a widget to the group and makes it the resizable widget.
• Fl_Widget +const + array () const
  Returns a pointer to the array of children.
• Fl_Group const + as_group () const FL OVERRIDE
• Fl_Group + as_group () FL OVERRIDE
  Returns an Fl_Group pointer if this widget is an Fl_Group.
• void begin ()
  Sets the current group so you can build the widget tree by just constructing the widgets.
• Fl_Widget * child (int n) const
  Returns array()[n].
• int children () const
  Returns how many child widgets the group has.
• void clear ()
  Deletes all child widgets from memory recursively.
• unsigned int clip_children ()
  Returns the current clipping mode.
• void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.
• virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.
• void end ()
  Exactly the same as current(this->parent()).
• int find (const Fl_Widget &o) const
  See int Fl_Group::find(const Fl_Widget *w) const.
• int find (const Fl_Widget *) const
  Searches the child array for the widget and returns the index.
• Fl_Group (int, int, int, int, const char * = 0)
  Creates a new Fl_Group widget using the given position, size, and label string.
• void focus (Fl_Widget *)
• void forms_end ()
  This is for forms compatibility only.
• int handle (int) FL OVERRIDE
  Handles the specified event.
• void init_sizes ()
  Resets the internal array of widget sizes and positions.
• void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.
• void insert (Fl_Widget &o, Fl_Widget * before)
  This does insert(w, find(before)).
• void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.
• void remove (Fl_Widget *)
  Removes the widget o from the group.
• void remove (int index)
  Removes the widget at index from the group but does not delete it.
• Fl_Widget * resizable () const
  Returns the group's resizable widget.
• void resizable (Fl_Widget &o)
  Sets the group's resizable widget.
• void resizable (Fl_Widget *o)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

• void resize (int, int, int, int) FL_OVERRIDE
  Resizes the Fl_Group widget and all of its children.

• virtual ~Fl_Group ()
  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

• void _clear_fullscreen ()
• void _set_fullscreen ()
• void activate ()
  Activates the widget.

• unsigned int active () const
  Returns whether the widget is active.

• int active_r () const
  Returns whether the widget and all of its parents are active.

• Fl_Align align () const
  Gets the label alignment.

• void align (Fl_Align alignment)
  Sets the label alignment.

• long argument () const
  Gets the current user data (long) argument that is passed to the callback function.

• void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window * as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const * as_gl_window () const

• virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window const * as_window () const

• void bind_deimage (Fl_Image *img)
  Sets the image to use as part of the widget label when in the inactive state.

• void bind_deimage (int f)
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

• void bind_image (Fl_Image *img)
  Sets the image to use as part of the widget label when in the active state.

• void bind_image (int f)
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

• Fl_Boxtype box () const
  Gets the box type of the widget.

• void box (Fl_Boxtype new_box)
  Sets the box type for the widget.

• Fl_Callback_p callback () const
  Gets the current callback function for the widget.

• void callback (Fl_Callback *cb)
  Sets the current callback function for the widget.

• void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)
  Sets the current callback function and managed user data for the widget.

• void callback (Fl_Callback *cb, void *p)
  Sets the current callback function and data for the widget.
• void callback (Fl_Callback *cb)
  Sets the current callback function for the widget.
• void callback (Fl_Callback1 *cb, long p=0)
  Sets the current callback function for the widget.
• unsigned int changed () const
  Checks if the widget value changed since the last callback.
• void clear_active ()
  Marks the widget as inactive without sending events or changing focus.
• void clear_changed ()
  Marks the value of the widget as unchanged.
• void clear_damage (uchar c=0)
  Clears or sets the damage flags.
• void clear_output ()
  Sets a widget to accept input.
• void clear_visible ()
  Hides the widget.
• void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.
• Fl_Color color () const
  Gets the background color of the widget.
• void color (Fl_Color bg)
  Sets the background color of the widget.
• void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.
• Fl_Color color2 () const
  For back compatibility only.
• void color2 (unsigned a)
  For back compatibility only.
• int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.
• void copy_label (const char *new_label)
  Sets the current label.
• void copy_tooltip (const char *text)
  Sets the current tooltip text.
• uchar damage () const
  Returns non-zero if draw() needs to be called.
• void damage (uchar c)
  Sets the damage bits for the widget.
• void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.
• int damage_resize (int, int, int, int)
  Internal use only.
• void deactivate ()
  Deactivates the widget.
• Fl_Image * deimage ()
  Gets the image that is used as part of the widget label when in the inactive state.
• const Fl_Image * deimage () const
  Gets the image that is used as part of the widget label when in the inactive state.
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label when in the inactive state.
• void deimage (Fl_Image *img)
Sets the image to use as part of the widget label when in the inactive state.

- int `deimage_bound()` const
  Returns whether the inactive image is managed by the widget.

- void `do_callback (Fl_Callback* reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with default arguments.

- void `do_callback (Fl_Widget* widget, long arg, Fl_Callback* reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.

- void `do_callback (Fl_Widget* widget, void* arg=0, Fl_Callback* reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.

- void `draw_label (int, int, int, int, Fl_Align) const`
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- int `h()` const
  Gets the widget height.

- virtual void `hide()`
  Makes a widget invisible.

- Fl_Image* `image()`
  Gets the image that is used as part of the widget label when in the active state.

- const Fl_Image* `image() const`
  Gets the image that is used as part of the widget label when in the active state.

- void `image (Fl_Image& img)`
  Sets the image to use as part of the widget label when in the active state.

- void `image (Fl_Image* img)`
  Sets the image to use as part of the widget label when in the active state.

- int `image_bound()` const
  Returns whether the image is managed by the widget.

- int `inside (const Fl_Widget* wgt) const`
  Checks if this widget is a child of `wgt`.

- int `is_label_copied()` const
  Returns whether the current label was assigned with `copy_label()`.

- const char* `label()` const
  Gets the current label text.

- void `label (const char* text)`
  Sets the current label pointer.

- void `label (Fl_Labeltype a, const char* b)`
  Shortcut to set the label text and type in one call.

- Fl_Color `labelcolor()` const
  Gets the label color.

- void `labelcolor (Fl_Color c)`
  Sets the label color.

- Fl_Font `labelfont()` const
  Gets the font to use.

- void `labelfont (Fl_Font f)`
  Sets the font to use.

- Fl_Fontsize `labelsiz()` const
  Gets the font size in pixels.

- void `labelsiz (Fl_Fontsize pix)`
  Sets the font size in pixels.

- Fl_Labeltype `labeltype()` const
  Gets the label type.

- void `labeltype (Fl_Labeltype a)`
  Sets the label type.
- `void measure_label (int &ww, int &hh) const`
  
  Sets width `ww` and height `hh` accordingly with the label size.

- `bool needs_keyboard () const`
  
  Returns whether this widget needs a keyboard.

- `void needs_keyboard (bool needs)`
  
  Sets whether this widget needs a keyboard.

- `unsigned int output () const`
  
  Returns if a widget is used for output only.

- `Fl_Group * parent () const`
  
  Returns a pointer to the parent widget.

- `void parent (Fl_Group *p)`
  
  Internal use only - “for hacks only”.

- `void position (int X, int Y)`
  
  Repositions the window or widget.

- `void redraw ()`
  
  Schedules the drawing of the widget.

- `void redraw_label ()`
  
  Schedules the drawing of the label.

- `Fl_Color selection_color () const`
  
  Gets the selection color.

- `void selection_color (Fl_Color a)`
  
  Sets the selection color.

- `void set_active ()`
  
  Marks the widget as active without sending events or changing focus.

- `void set_changed ()`
  
  Marks the value of the widget as changed.

- `void set_output ()`
  
  Sets a widget to output only.

- `void set_visible ()`
  
  Makes the widget visible.

- `void set_visible_focus ()`
  
  Enables keyboard focus navigation with this widget.

- `int shortcut_label () const`
  
  Returns whether the widget's label uses '&' to indicate shortcuts.

- `void shortcut_label (int value)`
  
  Sets whether the widget's label uses '&' to indicate shortcuts.

- `virtual void show ()`
  
  Makes a widget visible.

- `void size (int W, int H)`
  
  Changes the size of the widget.

- `int take_focus ()`
  
  Gives the widget the keyboard focus.

- `unsigned int takesevents () const`
  
  Returns if the widget is able to take events.

- `int testShortcut ()`
  
  Returns true if the widget's label contains the entered 'x' shortcut.

- `const char * tooltip () const`
  
  Gets the current tooltip text.

- `void tooltip (const char *text)`
  
  Sets the current tooltip text.

- `Fl_Window * top_window () const`
Returns a pointer to the top-level window for the widget.

- `Fl_Window * top_window_offset (int &xoff, int &yoff) const`
  Finds the x/y offset of the current widget relative to the top-level window.

- `uchar type () const`
  Gets the widget type.

- `void type (uchar t)`
  Sets the widget type.

- `int use_accents_menu ()`
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- `void * user_data () const`
  Gets the user data for this widget.

- `void user_data (Fl_Callback_User_Data *v, bool auto_free)`
  Sets the user data for this widget.

- `void user_data (void *v)`
  Sets the user data for this widget.

- `unsigned int visible () const`
  Returns whether a widget is visible.

- `unsigned int visible_focus () const`
  Checks whether this widget has a visible focus.

- `void visible_focus (int v)`
  Modifies keyboard focus navigation.

- `int visible_r () const`
  Returns whether a widget and all its parents are visible.

- `int w () const`
  Gets the widget width.

- `Fl_When when () const`
  Returns the conditions under which the callback is called.

- `void when (uchar i)`
  Sets the flags used to decide when a callback is called.

- `Fl_Window * window () const`
  Returns a pointer to the nearest parent window up the widget hierarchy.

- `int x () const`
  Gets the widget position in its window.

- `int y () const`
  Gets the widget position in its window.

- `virtual ~Fl_Widget ()`
  Destroys the widget.

Protected Member Functions

- `void draw () FL_OVERRIDE`
  Draws the wizard border and visible child.

Protected Member Functions inherited from Fl_Group

- `Fl_Rect * bounds ()`
  Returns the internal array of widget sizes and positions.

- `void draw_child (Fl_Widget &widget) const`
  Forces a child to redraw.

- `void draw_children ()`
  Draws all children of the group.

- `void draw_outside_label (const Fl_Widget &widget) const`
Parents normally call this to draw outside labels of child widgets.

- virtual int on_insert (Fl_Widget *, int)
  
  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)
  
  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)
  
  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()
  
  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const
  
  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void clear_flag (unsigned int c)
  
  Clears a flag in the flags mask.
- void draw_backdrop () const
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void draw_box () const
  
  Draws the widget box according its box style.
- void draw_box (Fl_Boxtype t, Fl_Color c) const
  
  Draws a box of type t, of color c at the widget's position and size.
- void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  
  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const
  
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  
  Draws a focus box for the widget at the given position and size.
- void draw_label () const
  
  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const
  
  Draws the label in an arbitrary bounding box.
- void FL_Widget (int x, int y, int w, int h, const char *label=0L)
  
  Creates a widget at the given position and size.
- unsigned int flags () const
  
  Gets the widget flags mask.
- void h (int v)
  
  Internal use only.
- void set_flag (unsigned int c)
  
  Sets a flag in the flags mask.
- void w (int v)
  
  Internal use only.
- void x (int v)
  
  Internal use only.
- void y (int v)
  
  Internal use only.
Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()
  
  Returns the currently active group.
- static void current (Fl_Group *g)
  
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)
  
  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)
  
  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)
  
  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
    INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
    FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
    OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
    MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
    GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCEPTS_MENU = 1<<19 ,
    NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
    AUTO_DELETE_USER_DATA = 1<<23 ,
    MAXIMIZED = 1<<24 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }

  flags possible values enumeration.

12.164.1 Detailed Description

This widget is based off the Fl_Tabs widget, but instead of displaying tabs it only changes "tabs" under program control.
Its primary purpose is to support "wizards" that step a user through configuration or troubleshooting tasks.
As with Fl_Tabs, wizard panes are composed of child (usually Fl_Group) widgets. Navigation buttons must be added separately.

12.164.2 Constructor & Destructor Documentation

12.164.2.1 Fl_Wizard()

Fl_Wizard::Fl_Wizard (  
  int xx,
  int yy,
  int ww,
  int hh,
  const char * l = 0 )

The constructor creates the Fl_Wizard widget at the specified position and size.
The inherited destructor destroys the widget and its children.
12.164.3 Member Function Documentation

12.164.3.1 draw()

void Fl_Wizard::draw (  
  void ) [protected], [virtual]  
Draws the wizard border and visible child.  
Reimplemented from Fl_Group.

12.164.3.2 next()

void Fl_Wizard::next ( )  
This method shows the next child of the wizard.  
If the last child is already visible, this function does nothing.

The documentation for this class was generated from the following files:

- Fl_Wizard.H
- Fl_Wizard.cxx

12.165 Fl_XBM_Image Class Reference

The Fl_XBM_Image class supports loading, caching, and drawing of X Bitmap (XBM) bitmap files.

#include <Fl_XBM_Image.H>

Inheritance diagram for Fl_XBM_Image:

```
Fl_XBM_Image
  Fl_Bitmap
  Fl_Image
```

Public Member Functions

- **Fl_XBM_Image** (const char ∗filename)  
  The constructor loads the named XBM file from the given name filename.

Public Member Functions inherited from Fl_Bitmap

- int cache_h ()
- int cache_w ()
- Fl_Image ∗copy () const  
  Creates a resized copy of the image.
- void draw (int X, int Y)  
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE  
  Draws the image to the current drawing surface with a bounding box.
- Fl_Bitmap (const char ∗bits, int bits_length, int W, int H)  
  The constructors create a new bitmap from the specified bitmap data.
- Fl_Bitmap (const char ∗bits, int W, int H)  
  The constructors create a new bitmap from the specified bitmap data.
- Fl_Bitmap (const uchar ∗bits, int bits_length, int W, int H)  
  The constructors create a new bitmap from the specified bitmap data.
- Fl_Bitmap (const uchar ∗bits, int W, int H)  
  The constructors create a new bitmap from the specified bitmap data.
The constructors create a new bitmap from the specified bitmap data.

- `void label (Fl_Menu_Item *m) FL_OVERRIDE`  
  This method is an obsolete way to set the image attribute of a menu item.

- `void label (Fl_Widget *w) FL_OVERRIDE`  
  This method is an obsolete way to set the image attribute of a widget or menu item.

- `void uncache () FL_OVERRIDE`  
  If the image has been cached for display, delete the cache data.

- `virtual ~Fl_Bitmap ()`  
  The destructor frees all memory and server resources that are used by the bitmap.

Public Member Functions inherited from `Fl_Image`

- `virtual class Fl_Shared_Image * as_shared_image ()`  
  Returns whether an image is an `Fl_Shared_Image` or not.

- `virtual void color_average (Fl_Color c, float i)`  
  The `color_average()` method averages the colors in the image with the provided FLTK color value.

- `Fl_Image * copy () const`  
  Creates a copy of the image in the same size.

- `int count () const`  
  Returns the number of data values associated with the image.

- `int d () const`  
  Returns the image depth.

- `const char * const * data () const`  
  Returns a pointer to the current image data array.

- `int data_h () const`  
  Returns the height of the image data.

- `int data_w () const`  
  Returns the width of the image data.

- `virtual void desaturate ()`  
  The `desaturate()` method converts an image to grayscale.

- `void draw (int X, int Y)`  
  Draws the image to the current drawing surface.

- `int fail () const`  
  Returns a value that is not 0 if there is currently no image available.

- `Fl_Image (int W, int H, int D)`  
  The constructor creates an empty image with the specified width, height, and depth.

- `int h () const`  
  Returns the current image drawing height in FLTK units.

- `void inactive ()`  
  The `inactive()` method calls `color_average(FL_BACKGROUND_COLOR, 0.33f)` to produce an image that appears grayed out.

- `int ld () const`  
  Returns the current line data size in bytes.

- `virtual void release ()`  
  Releases an `Fl_Image` - the same as 'delete this'.

- `virtual void scale (int width, int height, int proportional=1, int can_expand=0)`  
  Sets the drawing size of the image.

- `int w () const`  
  Returns the current image drawing width in FLTK units.

- `virtual ~Fl_Image ()`  
  The destructor is a virtual method that frees all memory used by the image.
Additional Inherited Members

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype define_FL_IMAGE_LABEL()
- static Fl_RGB_Scaling RGB_scaling()
  Returns the currently used RGB image scaling method.
- static void RGB_scaling(Fl_RGB_Scaling)
  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling scaling_algorithm()
  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm(Fl_RGB_Scaling algorithm)
  Sets what algorithm is used when resizing a source image to draw it.

Public Attributes inherited from Fl_Bitmap

- int alloc_array
  Non-zero if array points to bitmap data allocated internally.
- const uchar *array
  Pointer to raw bitmap data

Static Public Attributes inherited from Fl_Image

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_MEMORY_ACCESS = -4
- static const int ERR_NO_IMAGE = -1
- static bool register_images_done = false
  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_Image

- void d(int D)
  Sets the current image depth.
- void data(const char *const *p, int c)
  Sets the current data pointer and count of pointers in the array.
- void draw_empty(int X, int Y)
  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled(int X, int Y, int W, int H)
  Draw the image to the current drawing surface rescaled to a given width and height.
- void h(int H)
  Sets the height of the image data.
- void ld(int LD)
  Sets the current line data size in bytes.
- void w(int W)
  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void labeltype(const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void measure(const Fl_Label *lo, int &lw, int &lh)

12.165.1 Detailed Description

The Fl_XBM_Image class supports loading, caching, and drawing of X Bitmap (XBM) bitmap files.
12.165.2 Constructor & Destructor Documentation

12.165.2.1 Fl_XBM_Image()

Fl_XBM_Image::Fl_XBM_Image (const char ∗ name )
The constructor loads the named XBM file from the given name filename.
The destructor frees all memory and server resources that are used by the image.
The documentation for this class was generated from the following files:

- Fl_XBM_Image.H
- Fl_XBM_Image.cxx

12.166 Fl_XColor Struct Reference

Public Attributes

- unsigned char b
- unsigned char g
- unsigned char mapped
- unsigned long pixel
- unsigned char r

The documentation for this struct was generated from the following file:

- Fl_XColor.H

12.167 Fl_XPM_Image Class Reference

The Fl_XPM_Image class supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency.
#include <Fl_XPM_Image.H>

Inheritance diagram for Fl_XPM_Image:

```
Fl_Web
    Fl_XPM_Image
```

Public Member Functions

- Fl_XPM_Image (const char ∗ filename)
  The constructor loads the XPM image from the name filename.

Public Member Functions inherited from Fl_Pixmap

- int cache_h ()
- int cache_w ()
- void color_average (Fl_Color c, float i) FL_OVERRIDE
  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image ∗ copy () const
- Fl_Image ∗ copy (int W, int H) const FL_OVERRIDE
  Creates a resized copy of the image.
- void desaturate () FL_OVERRIDE

Generated by Doxygen
The desaturate() method converts an image to grayscale.

- void **draw** (int X, int Y)
- void **draw** (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE
  Draws the image to the current drawing surface with a bounding box.

- **Fl_Pixmap** (char ∗ const ∗ D)
  The constructors create a new pixmap from the specified XPM data.

- **Fl_Pixmap** (const char ∗ const ∗ D)
  The constructors create a new pixmap from the specified XPM data.

- **Fl_Pixmap** (const uchar ∗ const ∗ D)
  The constructors create a new pixmap from the specified XPM data.

- **Fl_Pixmap** (uchar ∗ const ∗ D)
  The constructors create a new pixmap from the specified XPM data.

- void **label** (Fl_Menu_Item ∗ m) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a menu item.

- void **label** (Fl_Widget ∗ w) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a widget or menu item.

- void **uncache** () FL_OVERRIDE
  If the image has been cached for display, delete the cache data.

- virtual ~**Fl_Pixmap** ()
  The destructor frees all memory and server resources that are used by the pixmap.

### Public Member Functions inherited from Fl_Image

- virtual class **Fl_Shared_Image** ∗ **as_shared_image** ()
  Returns whether an image is an Fl_Shared_Image or not.

- **Fl_Image** ∗ **copy** () const
  Creates a copy of the image in the same size.

- int **count** () const
  Returns the number of data values associated with the image.

- int **d** () const
  Returns the image depth.

- const char ∗ const ∗ **data** () const
  Returns a pointer to the current image data array.

- int **data_h** () const
  Returns the height of the image data.

- int **data_w** () const
  Returns the width of the image data.

- void **draw** (int X, int Y)
  Draws the image to the current drawing surface.

- int **fail** () const
  Returns a value that is not 0 if there is currently no image available.

- **Fl_Image** (int W, int H, int D)
  The constructor creates an empty image with the specified width, height, and depth.

- int **h** () const
  Returns the current image drawing height in FLTK units.

- void **inactive** ()
  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

- int **ld** () const
  Returns the current line data size in bytes.

- virtual void **release** ()
Releases an `Fl_Image` - the same as `delete this`.

- virtual void `scale` (int width, int height, int proportional=1, int can_expand=0)
  
  Sets the drawing size of the image.

- int `w` () const

  Returns the current image drawing width in FLTK units.

- virtual `~Fl_Image` ()

  The destructor is a virtual method that frees all memory used by the image.

### Additional Inherited Members

#### Static Public Member Functions inherited from `Fl_Image`

- static `Fl_Labeltype define_FL_IMAGE_LABEL`

- static `Fl_RGB_Scaling RGB_scaling`

  Returns the currently used RGB image scaling method.

- static void `RGB_scaling (Fl_RGB_Scaling)`

  Sets the RGB image scaling method used for `copy(int, int)`.

- static `Fl_RGB_Scaling scaling_algorithm`

  Gets what algorithm is used when resizing a source image to draw it.

- static void `scaling_algorithm (Fl_RGB_Scaling algorithm)`

  Sets what algorithm is used when resizing a source image to draw it.

#### Public Attributes inherited from `Fl_Pixmap`

- int `alloc_data`

#### Static Public Attributes inherited from `Fl_Image`

- static const int `ERR_FILE_ACCESS = -2`

- static const int `ERR_FORMAT = -3`

- static const int `ERR_MEMORY_ACCESS = -4`

- static const int `ERR_NO_IMAGE = -1`

- static bool `register_images_done = false`

  True after `fl_register_images()` was called, false before.

#### Protected Member Functions inherited from `Fl_Pixmap`

- void `measure` ()

#### Protected Member Functions inherited from `Fl_Image`

- void `d` (int D)

  Sets the current image depth.

- void `data` (const char *const *p, int c)

  Sets the current data pointer and count of pointers in the array.

- void `draw_empty` (int X, int Y)

  The protected method `draw_empty()` draws a box with an `X` in it.

- int `draw_scaled` (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.

- void `h` (int H)

  Sets the height of the image data.

- void `ld` (int LD)

  Sets the current line data size in bytes.

- void `w` (int W)

  Sets the width of the image data.
Static Protected Member Functions inherited from Fl_image

- static void labeltype (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void measure (const Fl_Label *lo, int &lw, int &lh)

12.167.1 Detailed Description

The Fl_XPM_Image class supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency.

12.167.2 Constructor & Destructor Documentation

12.167.2.1 Fl_XPM_Image()

Fl_XPM_Image::Fl_XPM_Image (const char * name )

The constructor loads the XPM image from the name filename.
The destructor frees all memory and server resources that are used by the image.
The documentation for this class was generated from the following files:
- Fl_XPM_Image.H
- Fl_XPM_Image.cxx

12.168 Fl_GIF_Image::GIF_FRAME Struct Reference

Classes

- struct CPAL

Public Member Functions

- void colors (int nclrs, int bg, int tp)
- void disposal (int mode, int delay)
- GIF_FRAME (int frame, int W, int H, int fx, int fy, int fw, int fh, uchar * data)
- GIF_FRAME (int frame, uchar * data)

Public Attributes

- int bkgd
- const uchar * bptr
- int clrs
- const struct Fl_GIF_Image::GIF_FRAME::CPAL * cpal
- int delay
- int dispose
- int h
- int height
- int ifrm
- int trans
- int w
- int width
- int x
- int y

The documentation for this struct was generated from the following file:
- Fl_GIF_Image.H
12.169  gtk_shell1_listener Struct Reference

Public Attributes

- `void(* capabilities)(void *data, struct gtk_shell1 *gtk_shell1, uint32_t capabilities)`

The documentation for this struct was generated from the following file:

- gtk-shell-client-protocol.h

12.170  gtk_surface1_listener Struct Reference

Public Attributes

- `void(* configure)(void *data, struct gtk_surface1 *gtk_surface1, struct wl_array *states)`
- `void(* configure_edges)(void *data, struct gtk_surface1 *gtk_surface1, struct wl_array *constraints)`

12.170.1  Member Data Documentation

12.170.1.1  configure_edges

`void(* gtk_surface1_listener::configure_edges)(void *data, struct gtk_surface1 *gtk_surface1, struct wl_array *constraints)`

Since

2

The documentation for this struct was generated from the following file:

- gtk-shell-client-protocol.h

12.171  Fl_ICO_Image::IconDirEntry Struct Reference

Windows ICONDIRENTRY structure

#include <Fl_ICO_Image.H>

Public Attributes

- `int bColorCount`
  
  Number of colors (0 if 8bpp)
- `int bHeight`
  
  Image height.
- `int bReserved`
  
  Reserved.
- `int bWidth`
  
  Image width.
- `int dwBytesInRes`
  
  Resource size in bytes.
- `int dwImageOffset`
  
  Offset to the image.
- `int wBitCount`
  
  Bits per pixel.
- `int wPlanes`
  
  Color Planes.
12.171.1 Detailed Description

Windows ICONDIRENTRY structure

The documentation for this struct was generated from the following file:

- Fl_ICO_Image.H

12.172 Fl_Text_Editor::Key_Binding Struct Reference

Simple linked list item associating a key/state to a function.

```
#include <Fl_Text_Editor.H>
```

**Public Attributes**

- `Key_Func` function
  - associated function
- `int key`
  - the key pressed
- `Key_Binding * next`
  - next key binding in the list
- `int state`
  - the state of key modifiers

12.172.1 Detailed Description

Simple linked list item associating a key/state to a function.

The documentation for this struct was generated from the following file:

- Fl_Text_Editor.H

12.173 Fl_Terminal::Margin Class Reference

**Public Member Functions**

- `void bottom (int val)`
- `int bottom (void) const`
- `void left (int val)`
- `int left (void) const`
- `void right (int val)`
- `int right (void) const`
- `void top (int val)`
- `int top (void) const`

The documentation for this class was generated from the following file:

- Fl_Terminal.H

12.174 Fl_Preferences::Name Class Reference

'Name' provides a simple method to create numerical or more complex procedural names for entries and groups on the fly.

```
#include <Fl_Preferences.H>
```
Public Member Functions

- **Name** (const char ∗format,...)
  Creates a group name or entry name on the fly.
- **Name** (unsigned int n)
  Creates a group name or entry name on the fly.
- **operator const char ∗()**
  Return the Name as a "C" string.

12.174.1 Detailed Description

`Name` provides a simple method to create numerical or more complex procedural names for entries and groups on the fly.

Example: prefs.set(Fl_Preferences::Name("File%d",i),file[i]);
See test/preferences.cxx as a sample for writing arrays into preferences.

`Name` is actually implemented as a class inside Fl_Preferences. It casts into const char* and gets automatically destroyed after the enclosing call ends.

12.174.2 Constructor & Destructor Documentation

12.174.2.1 Name() [1/2]

Fl_Preferences::Name::Name (unsigned int n)

Creates a group name or entry name on the fly.
This version creates a simple unsigned integer as an entry name.

int n, i;
Fl_Preferences prev( appPrefs, "PreviousFiles" );
prev.get( "n", 0 );
for ( i=0; i<n; i++ )
  prev.get( Fl_Preferences::Name(i), prevFile[i], "" );

12.174.2.2 Name() [2/2]

Fl_Preferences::Name::Name (const char ∗format, ...)

Creates a group name or entry name on the fly.
This version creates entry names as in 'printf'.

int n, i;
Fl_Preferences prefs( USER, "matthiasm.com", "test" );
prefs.get( "nFiles", 0 );
for ( i=0; i<n; i++ )
  prefs.get( Fl_Preferences::Name("File%d", i), prevFile[i], "" );

The documentation for this class was generated from the following files:

- Fl_Preferences.H
- Fl_Preferences.cxx

12.175 Fl_Preferences::Node Class Reference

Public Member Functions

- **void add** (const char ∗line)
- **Node ∗addChild** (const char ∗path)
- **const char ∗child** (int ix)
- **Node ∗childNode** (int ix)
- **void clearDirtyFlags** ()
- **void deleteAllChildren** ()
- **void deleteAllEntries** ()
- **char deleteEntry** (const char ∗name)
• char dirty ()
• Entry & entry (int i)
• Node * find (const char *path)
• RootNode * findRoot ()
const char * get (const char *name)
• int getEntry (const char *name)
• const char * name ()
• int nChildren ()
• int nEntry ()
• Node (const char *path)
• Node * parent ()
• const char * path ()
• char remove ()
• Node * search (const char *path, int offset=0)
• void set (const char *line)
• void set (const char *name, const char *value)
• void setParent (Node *parent)
• void setRoot (RootNode *r)
• int write (FILE *f)

Static Public Attributes

• static int lastEntrySet = -1

The documentation for this class was generated from the following files:

• Fl_Preferences.H
• Fl_Preferences.cxx

12.176 Fl_Paged_Device::page_format Struct Reference

width, height and name of a page format
#include <Fl_Paged_Device.H>

Public Attributes

• int height
  height in points
• const char * name
  format name
• int width
  width in points

12.176.1 Detailed Description

width, height and name of a page format
The documentation for this struct was generated from the following file:

• Fl_Paged_Device.H
12.177 Fl_Terminal::PartialUtf8Buf Class Reference

Public Member Functions

- bool append (const char *p, int len)
- const char *buf (void) const
- int buflen (void) const
- void clear (void)
- bool is_complete (void) const
- bool is_continuation (char c)

The documentation for this class was generated from the following file:

- Fl_Terminal.H

12.178 Fl_Terminal::RingBuffer Class Reference

Public Member Functions

- void change_disp_cols (int dcols, const CharStyle &style)
- void change_disp_rows (int drows, const CharStyle &style)
- void clear (void)
- void clear_disp_rows (int sdrow, int edrow, const CharStyle &style)
- void clear_hist (void)
- void create (int drows, int dcols, int hrows)
- int disp_cols (void) const
- int disp_erow (void) const
- void disp_rows (int val)
- int disp_rows (void) const
- int disp_srow (void) const
- int hist_cols (void) const
- int hist_erow (void) const
- void hist_rows (int val)
- int hist_rows (void) const
- int hist_srow (void) const
- void hist_use (int val)
- int hist_use (void) const
- int hist_use_srow (void) const
- bool is_disp_ring_row (int grow) const
- bool is_hist_ring_row (int grow) const
- void move_disp_row (int src_row, int dst_row)
- int offset (void) const
- void offset_adjust (int rows)
- void resize (int drows, int dcols, int hrows, const CharStyle &style)
- Utf8Char *ring_chars (void)
- Utf8Char *ring_chars (void) const
- int ring_cols (void) const
- int ring_erow (void) const
- int ring_rows (void) const
- int ring_srow (void) const
- RingBuffer (int drows, int dcols, int hrows)
- void scroll (int rows, const CharStyle &style)
- Utf8Char *u8c_disp_row (int drow)
- const Utf8Char *u8c_disp_row (int drow) const
- Utf8Char *u8c_hist_row (int hrow)
- const Utf8Char *u8c_hist_row (int hrow) const
The documentation for this class was generated from the following files:

* FL_Terminal.H
* FL_Terminal.cxx

### 12.179 FI_Preferences::RootNode Class Reference

#### Public Member Functions

- char * filename ()
- char getPath (char *path, int pathlen)
- int read ()
- Root root ()
- RootNode (FI_Preferences *)
- RootNode (FI_Preferences *, const char *path, const char *vendor, const char *application, Root flags)
- RootNode (FI_Preferences *, Root root, const char *vendor, const char *application)
- int write ()

The documentation for this class was generated from the following files:

* FI_Preferences.H
* FI_Preferences.cxx

### 12.180 FI_Scroll::ScrollInfo Struct Reference

Structure to manage scrollbar and widget interior sizes.

```
#include <Fl_Scroll.H>
```

#### Public Attributes

- FL_Region_LRTB child
  - child bounding box: left/right/top/bottom
- int hneeded
  - horizontal scrollbar visibility
- FI_Scrollbar_Data hscroll
  - horizontal scrollbar region + values
- FL_Region_XYWH innerbox
  - widget's inner box, excluding scrollbars
- FL_Region_XYWH innerchild
  - widget's inner box, including scrollbars
- int scrollsize
  - the effective scrollbar thickness (local or global)
- int vneeded
  - vertical scrollbar visibility
- FI_Scrollbar_Data vscroll
  - vertical scrollbar region + values
12.180.1 Detailed Description

Structure to manage scrollbar and widget interior sizes. This is filled out by `recalc_scrollbars()` for use in calculations that need to know the visible scroll area size, etc.

Version

1.3.3

The documentation for this struct was generated from the following file:

- Fl_Scroll.H

12.181 Fl_Terminal::Selection Class Reference

Public Member Functions

- bool `clear` (void)
- bool `dragged_off` (int row, int col)
- int `ecol` (void) const
- void `end` (void)
- int `erow` (void) const
- bool `extend` (int row, int col)
- bool `get_selection` (int &srow, int &scol, int &erow, int &ecol) const
  
  Return selection start/end.
- bool `is_selection` (void) const
- void `push_clear` ()
- void `push_rowcol` (int row, int col)
- int `scol` (void) const
- void `scroll` (int nrows)
- void `select` (int srow, int scol, int erow, int ecol)
- void `selectionbgcolor` (Fl_Color val)
- Fl_Color `selectionbgcolor` (void) const
- void `selectionfgcolor` (Fl_Color val)
- Fl_Color `selectionfgcolor` (void) const
- int `srow` (void) const
- bool `start` (int row, int col)
- void `start_push` ()
- int `state` (void) const

12.181.1 Member Function Documentation

12.181.1.1 get_selection()

```cpp
bool Fl_Terminal::Selection::get_selection (  
    int & srow,
    int & scol,
    int & erow,
    int & ecol ) const
```

Return selection start/end.

Ensure (start < end) to allow walking 'forward' thru selection, left-to-right, top-to-bottom.

Returns:

- true – valid selection values returned
- false – no selection was made, returned values undefined

The documentation for this class was generated from the following files:

- Fl_Terminal.H
- Fl_Terminal.cxx

Generated by Doxygen
12.182  **Fl_Tile::Size_Range Struct Reference**

Public Attributes

- int maxh
- int maxw
- int minh
- int minw

The documentation for this struct was generated from the following file:

- Fl_Tile.H

12.183  **Fl_Text_Display::Style_Table_Entry Struct Reference**

This structure associates the color, font, and font size of a string to draw with an attribute mask matching attr.

#include <Fl_Text_Display.H>

Public Attributes

- unsigned attr
  further attributes for the text style (see ATTR_BGCOLOR, etc.)
- Fl_Color bgcolor
  text background color if ATTR_BGCOLOR or ATTR_BGCOLOR_EXT is set
- Fl_Color color
  text color
- Fl_Font font
  text font
- Fl_Fontsize size
  text font size

12.183.1 Detailed Description

This structure associates the color, font, and font size of a string to draw with an attribute mask matching attr. There must be one entry for each style that can be used in an Fl_Text_Display for displaying text. The style table is an array of struct Style_Table_Entry. The style table is associated with an Fl_Text_Display by using Fl_Text_Display::highlight_data().

See also

Fl_Text_Display::highlight_data()

The documentation for this struct was generated from the following file:

- Fl_Text_Display.H

12.184  **Fl_Terminal::Utf8Char Class Reference**

Public Member Functions

- Fl_Color attr_bg_color (const Fl_Widget *grp) const
- Fl_Color attr_color (Fl_Color col, const Fl_Widget *grp) const
- Fl_Color attr_fg_color (const Fl_Widget *grp) const
- uchar attrib (void) const
- Fl_Color bgcolor (void) const
- uchar charflags (void) const
- void clear (const CharStyle &style)
- Fl_Color fgcolor (void) const
• void **fl_font_set** (const CharStyle &style) const
• bool **is_char** (char c) const
• int **length** (void) const
• int **max_utf8** () const
• Utf8Char & **operator=** (const Utf8Char &o)
• double **pwidth** (void) const
• int **pwidth_int** (void) const
• void **show_char** (void) const
• void **show_char_info** (void) const
• void **text_ascii** (char c, const CharStyle &style)
• void **text_utf8** (const char *text, int len, const CharStyle &style)
• const char * **text_utf8** (void) const
• Utf8Char (const Utf8Char &o)

The documentation for this class was generated from the following files:

• FL_Terminal.H
• FL_Terminal.hxx
Chapter 13

File Documentation

13.1 Enumerations.H File Reference

This file contains type definitions and general enumerations.
#include <FL/fl_config.h>
#include "Fl_Export.H"
#include "fl_types.h"
#include <FL/platform_types.h>

Macros

• #define FL_IMAGE_WITH_ALPHA 0x40000000

Version Numbers

FLTK defines some constants to help the programmer to find out, for which FLTK version a program is compiled.
The following constants are defined:

• #define FL_ABI_VERSION FL_API_VERSION
  The FLTK ABI (Application Binary Interface) version number as an int.
• #define FL_API_VERSION (FL_MAJOR_VERSION*10000 + FL_MINOR_VERSION*100 + FL_PATCH_VERSION)
  The FLTK API version number as an int.
• #define FL_MAJOR_VERSION 1
  The major release version of this FLTK library.
• #define FL_MINOR_VERSION 4
  The minor release version for this library.
• #define FL_PATCH_VERSION 0
  The patch version for this library.
• #define FL_VERSION
  The FLTK version number as a double.

Names of Non-ASCII keys and mouse buttons

The following constants define the names of non-ASCII keys on the keyboard and of mouse buttons for FL←KEYBOARD and FL_SHORTCUT events.

See also

Fl::event_key() and Fl::get_key(int) (use ASCII letters for all other keys):

• #define FL_Alt_L 0xffe9
  The left alt key.
• #define FL_Alt_R 0xffea
  The right alt key.
• #define FL_Back 0xEF26
  Like back on a browser.
• \#define FL_BackSpace 0xff08
  The backspace key.
• \#define FL_Button 0xfee8
  A mouse button; use FL_Button + n for mouse button n.
• \#define FL_Caps_Lock 0xffe5
  The caps lock key.
• \#define FL_Control_L 0xffe3
  The lefthand control key.
• \#define FL_Control_R 0xffe4
  The righthand control key.
• \#define FL_Delete 0xffff
  The delete key.
• \#define FL_Down 0xff54
  The down arrow key.
• \#define FL_Eisu 0xff2f
  The Eisu key of JIS keyboards.
• \#define FL_End 0xff57
  The end key.
• \#define FL_Enter 0xff0d
  The enter key.
• \#define FL_Escape 0xff1b
  The escape key.
• \#define FL_F 0xffbd
  One of the function keys; use FL_F + n for function key n.
• \#define FL_F_Last 0xffe0
  The last function key; use to range-check function keys.
• \#define FL_Favorites 0xEF30
  Show favorite locations.
• \#define FL_Forward 0xEF27
  Like forward on a browser.
• \#define FL_Help 0xff68
  The 'help' key on Mac keyboards.
• \#define FL_Home 0xff50
  The home key.
• \#define FL_Home_Page 0xEF18
  Display user's home page.
• \#define FL_Insert 0xff63
  The insert key.
• \#define FL_Iso_Key 0xff0c
  The additional key of ISO keyboards.
• \#define FL_JIS_Underscore 0xff31
  The underscore key of JIS keyboards.
• \#define FL_Kana 0xff2e
  The Kana key of JIS keyboards.
• \#define FL_KP 0xff80
  One of the keypad numbers; use FL_KP + 'n' for digit n.
• \#define FL_KP_Enter 0xff8d
  The enter key on the keypad, same as FL_KP+'r'.
• \#define FL_KP_Last 0xffbd
  The last keypad key; use to range-check keypad.
• \#define FL_Left 0xff51
  The left arrow key.
• \#define FL_Mail 0xEF19
  Invoke user's mail program.
• \#define FL_Media_Next 0xEF17
  Next track.
• \#define FL_Media_Play 0xEF14
  Start playing of audio.
• \#define FL_Media_Prev 0xEF16
# Enumerations.H

**File Reference 1967**

- **#define FL_Media_Stop** 0xEF15
  - Stop playing audio.
- **#define FL_Menu** 0xff67
  - The menu key.
- **#define FL_Meta_L** 0xffe7
  - The left meta/Windows key.
- **#define FL_Meta_R** 0xffe8
  - The right meta/Windows key.
- **#define FL_Num_Lock** 0xff7f
  - The num lock key.
- **#define FL_Page_Down** 0xff56
  - The page-down key.
- **#define FL_Page_Up** 0xff55
  - The page-up key.
- **#define FL_Pause** 0xff13
  - The pause key.
- **#define FL_Print** 0xff61
  - The print (or print-screen) key.
- **#define FL_Refresh** 0xEF29
  - Refresh the page.
- **#define FL_Right** 0xff53
  - The right arrow key.
- **#define FL_Scroll_Lock** 0xff14
  - The scroll lock key.
- **#define FL_Search** 0xEF1B
  - Search.
- **#define FL_Shift_L** 0xffe1
  - The lefthand shift key.
- **#define FL_Shift_R** 0xffe2
  - The righthand shift key.
- **#define FL_Sleep** 0xEF2F
  - Put system to sleep.
- **#define FL_Stop** 0xEF28
  - Stop current operation.
- **#define FL_Tab** 0xff09
  - The tab key.
- **#define FL_Up** 0xff52
  - The up arrow key.
- **#define FL_Volume_Down** 0xEF11
  - Volume control down.
- **#define FL_Volume_Mute** 0xEF12
  - Mute sound from the system.
- **#define FL_Volume_Up** 0xEF13
  - Volume control up.
- **#define FL_Yen** 0xff30
  - The Yen key of JIS keyboards.

**Mouse Buttons**

These constants define the button numbers for FL_PUSH and FL_RELEASE events.

See also

`Fl::event_button()`

- **#define FL_LEFT_MOUSE** 1
  - The left mouse button.
- **#define FL_MIDDLE_MOUSE** 2
  - The middle mouse button.
- **#define FL_RIGHT_MOUSE** 3

Generated by Doxygen
The right mouse button.

### Event States

The following constants define bits in the Fl::event_state() value.

- ```#define FL_ALT 0x00080000```
  One of the alt keys is down.
- ```#define FL_BUTTON(n) (0x00800000<<(n))```
  Mouse button n (n > 0) is pushed.
- ```#define FL_BUTTON1 0x01000000```
  Mouse button 1 is pushed.
- ```#define FL_BUTTON2 0x02000000```
  Mouse button 2 is pushed.
- ```#define FL_BUTTON3 0x04000000```
  Mouse button 3 is pushed.
- ```#define FL_BUTTONS 0x7f000000```
  Any mouse button is pushed.
- ```#define FL_CAPS_LOCK 0x00020000```
  The caps lock is on.
- ```#define FL_CTRL 0x00040000```
  One of the ctrl keys is down.
- ```#define FL_KEY_MASK 0x0000ffff```
  All keys are 16 bit for now.
- ```#define FL_META 0x00400000```
  One of the meta/Windows keys is down.
- ```#define FL_NUM_LOCK 0x00100000```
  The num lock is on.
- ```#define FL_SCROLL_LOCK 0x00800000```
  The scroll lock is on.
- ```#define FL_SHIFT 0x00010000```
  One of the shift keys is down.

### Typedefs

- ```typedef int Fl_Fontsize```
  Size of a font in pixels.

### Enumerations

- ```enum { FL_READ = 1 , FL_WRITE = 4 , FL_EXCEPT = 8 }```
  FD "when" conditions.
- ```enum Fl_Arrow_Type { FL_ARROW_SINGLE = 0x01 , FL_ARROW_DOUBLE = 0x02 , FL_ARROW_CHOICE = 0x03 , FL_ARROW_RETURN = 0x04 }```
  Arrow types define the type of arrow drawing function.
- ```enum Fl_Damage {
  FL_DAMAGE_CHILD = 0x01 , FL_DAMAGE_EXPOSE = 0x02 , FL_DAMAGE_SCROLL = 0x04 ,
  FL_DAMAGE_OVERLAY = 0x08 ,
  FL_DAMAGE_USER1 = 0x10 , FL_DAMAGE_USER2 = 0x20 , FL_DAMAGE_ALL = 0x80 }
```
  Damage masks.
- ```enum Fl_Event {
  FL_NO_EVENT = 0 , FL_PUSH = 1 , FL_RELEASE = 2 , FL_ENTER = 3 ,
  FL_LEAVE = 4 , FL_DRAG = 5 , FL_FOCUS = 6 , FL_UNFOCUS = 7 ,
  FL_KEYDOWN = 8 , FL_KEYBOARD = 9 , FL_KEYUP = 9 , FL_CLOSE = 10 ,
  FL_MOVE = 11 , FL_SHORTCUT = 12 , FL_DEACTIVATE = 13 , FL_ACTIVATE = 14 ,
  FL_HIDE = 15 , FL_SHOW = 16 , FL_PASTE = 17 , FL_SELECTIONCLEAR = 18 ,
  FL_MOUSEWHEEL = 19 , FL_DND_ENTER = 20 , FL_DND_DRAG = 21 , FL_DND_LEAVE = 22 ,
  FL_DND_RELEASE = 23 , FL_SCREEN_CONFIGURATION_CHANGED = 24 , FL_FULLSCREEN = 25 ,
  FL_ZOOM_GESTURE = 26 ,
  FL_ZOOM_EVENT = 27 }
```
Every time a user moves the mouse pointer, clicks a button, or presses a key, an event is generated and sent to your application.

- `enum Fl_Labeltype {
  FL_NORMAL_LABEL = 0, FL_NO_LABEL, FL_SHADOW_LABEL, FL_ENGRAVED_LABEL,
  FL_EMBOSSED_LABEL, FL_MULTI_LABEL, FL_ICON_LABEL, FL_IMAGE_LABEL,
  FL_FREE_LABELTYPE
}

The `labeltype()` method sets the type of the label.

- `enum Fl_Mode {
  FL_RGB = 0, FL_INDEX = 1, FL_SINGLE = 0, FL_DOUBLE = 2,
  FL_ACCUM = 4, FL_ALPHA = 8, FL_DEPTH = 16, FL_STENCIL = 32,
  FL_RGB8 = 64, FL_MULTISAMPLE = 128, FL_STEREO = 256, FL_FAKE_SINGLE = 512,
  FL_OPENGL3 = 1024
}

These constants match `Glut`.

- `enum Fl_Orientation {
  FL_ORIENTATION_NONE = 0x00, FL_ORIENTATION_RIGHT = 0x00, FL_ORIENTATION_NE = 0x01,
  FL_ORIENTATION_UP = 0x02, FL_ORIENTATION_NW = 0x03, FL_ORIENTATION_LEFT = 0x04,
  FL_ORIENTATION_SW = 0x05, FL_ORIENTATION_DOWN = 0x06,
  FL_ORIENTATION_SE = 0x07
}

`Fl_Orientation` describes the orientation of a GUI element.

- `enum Fl_When {
  FL_WHEN_NEVER = 0, FL_WHEN_CHANGED = 1, FL_WHEN_NOT_CHANGED = 2, FL_WHEN_RELEASE = 4,
  FL_WHEN_RELEASE_ALWAYS = 6, FL_WHEN_ENTER_KEY = 8, FL_WHEN_ENTER_KEY_ALWAYS = 10,
  FL_WHEN_ENTER_KEY_CHANGED = 11, FL_WHEN_CLOSED = 16
}

These constants determine when a callback is performed.

- `enum Fl_Callback_Reason {
  FL_REASON_UNKNOWN = 0, FL_REASON_SELECTED, FL_REASON_DESELECTED, FL_REASON_RESELECTED,
  FL_REASON_OPENED, FL_REASON_CLOSED, FL_REASON_DRAGGED, FL_REASON_CANCELLED,
  FL_REASON_CHANGED, FL_REASON_GOT_FOCUS, FL_REASON_LOST_FOCUS, FL_REASON_RELEASED,
  FL_REASON_ENTER_KEY, FL_REASON_USER = 32
}

These constants describe why a callback is performed.

- `enum Fl_Cursor {
  FL_CURSOR_DEFAULT = 0, FL_CURSOR_ARROW = 35, FL_CURSOR_CROSS = 66,
  FL_CURSOR_WAIT = 76, FL_CURSOR_INSERT = 77, FL_CURSOR_HAND = 31, FL_CURSOR_HELP = 47,
  FL_CURSOR_MOVE = 27, FL_CURSOR_NS = 78, FL_CURSOR_WE = 79, FL_CURSOR_NWSE = 80,
  FL_CURSOR_NESW = 81, FL_CURSOR_N = 70, FL_CURSOR_NE = 69, FL_CURSOR_E = 49,
  FL_CURSOR_SE = 8, FL_CURSOR_S = 9, FL_CURSOR_SW = 7, FL_CURSOR_W = 36,
  FL_CURSOR_NW = 68, FL_CURSOR_NONE = 255
}

The following constants define the mouse cursors that are available in FLTK.
Variables

- **FL_Fontsize FL_NORMAL_SIZE**
  
  normal font size

Box Types

FLTK standard box types

This enum defines the standard box types included with FLTK.

**Note**

The documented enum `Fl_Boxtype` contains some values (names) with leading underscores, e.g. `FL_SHADOW_BOX`. This is due to technical reasons - please use the same values (names) without the leading underscore in your code! Enum values with leading underscores are reserved for internal use and subject to change without notice!

FL_NO_BOX means nothing is drawn at all, so whatever is already on the screen remains. The FL_..._FRAME types only draw their edges, leaving the interior unchanged. The blue color in the image below is the area that is not drawn by the frame types.

![Figure 13.1 FLTK Standard Box Types](image)

**Figure 13.1 FLTK Standard Box Types**

**Note**

Not all box types are depicted in the figure above. See enum `Fl_Boxtype` below for the complete list of box types.

**See also**

`Fl::get_system_colors()`

- **Fl_Boxtype fl_box (Fl_Boxtype b)**

  Get the filled version of a frame.
• enum FL_Boxtype {
    FL_NO_BOX = 0, FL_FLAT_BOX, FL_UP_BOX, FL_DOWN_BOX,
    FL_UP_FRAME, FL_DOWN_FRAME, FL_THIN_UP_BOX, FL_THIN_DOWN_BOX,
    FL_THIN_UP_FRAME, FL_THIN_DOWN_FRAME, FL_ENGRAVED_BOX, FL_EMBOSSED_BOX,
    FL_ENGRAVED_FRAME, FL_EMBOSSED_FRAME, FL_BORDER_BOX, FL_SHADOW_BOX,
    FL_BORDER_FRAME, FL_SHADOW_FRAME, FL_ROUNDED_BOX, FL_SHADOW_BOX,
    FL_ROUNDED_FRAME, FL_RFLAT_BOX, FL_ROUND_UP_BOX, FL_ROUND_DOWN_BOX,
    FL_DIAMOND_UP_BOX, FL_DIAMOND_DOWN_BOX, FL_OVAL_BOX, FL_OSHADOW_BOX,
    FL_OVAL_FRAME, FL_OFLAT_BOX, FL_PLASTIC_UP_BOX, FL_PLASTIC_DOWN_BOX,
    FL_PLASTIC_UP_FRAME, FL_PLASTIC_DOWN_FRAME, FL_PLASTIC_THIN_UP_BOX,
    FL_PLASTIC_THIN_DOWN_BOX,
    FL_PLASTIC_ROUND_UP_BOX, FL_PLASTIC_ROUND_DOWN_BOX, FL_GTK_UP_BOX,
    FL_GTK_DOWN_BOX,
    FL_GTK_UP_FRAME, FL_GTK_DOWN_FRAME, FL_GTK_THIN_UP_BOX, FL_GTK_THIN_DOWN_BOX,
    FL_GTK_THIN_UP_FRAME, FL_GTK_THIN_DOWN_FRAME, FL_GTK_ROUND_UP_BOX,
    FL_GTK_ROUND_DOWN_BOX, FL_GLEAM_UP_BOX, FL_GLEAM_DOWN_BOX, FL_GLEAM_UP_FRAME, FL_GLEAM_DOWN_FRAME,
    FL_GLEAM_THIN_UP_BOX, FL_GLEAM_THIN_DOWN_BOX, FL_GLEAM_ROUND_UP_BOX,
    FL_GLEAM_ROUND_DOWN_BOX, FL_OXY_UP_BOX, FL_OXY_DOWN_BOX, FL_OXY_UP_FRAME, FL_OXY_DOWN_FRAME,
    FL_OXY_THIN_UP_BOX, FL_OXY_THIN_DOWN_BOX, FL_OXY_THIN_UP_FRAME, FL_OXY_THIN_DOWN_FRAME,
    FL_OXY_BUTTON_UP_BOX, FL_OXY_BUTTON_DOWN_BOX,
    FL_FREE_BOXTYPE}

    FLTK standard box types.
• #define FL_CIRCLE_BOX FL_ROUND_DOWN_BOX
• #define FL_DIAMOND_BOX FL_DIAMOND_DOWN_BOX
• FL_Boxtype fl_down (FL_Boxtype b)
  Get the "pressed" or "down" version of a box.
• FL_Boxtype fl_frame (FL_Boxtype b)
  Get the unfilled, frame only version of a box.
• #define FL_FRAME FL_ENGRAVED_FRAME
• #define FL_FRAME_BOX FL_ENGRAVED_BOX
• FL_Labeltype fl_define_FL_EMBOSSED_LABEL ()
  Initializes the internal table entry for FL_EMBOSSED_LABEL and returns its internal value.
• FL_Labeltype fl_define_FL_ENGRAVED_LABEL ()
  Initializes the internal table entry for FL_ENGRAVED_LABEL and returns its internal value.
• FL_Labeltype fl_define_FL_ICON_LABEL ()
  Initializes the internal table entry for FL_ICON_LABEL and returns its internal value.
• FL_Labeltype fl_define_FL_IMAGE_LABEL ()
  Initializes the internal table entry for FL_IMAGE_LABEL and returns its internal value.
• FL_Labeltype fl_define_FL_MULTI_LABEL ()
  Initializes the internal table entry for FL_MULTI_LABEL and returns its internal value.
• FL_Labeltype fl_define_FL_SHADOW_LABEL ()
  Initializes the internal table entry for FL_SHADOW_LABEL and returns its internal value.
• #define FL_EMBOSSED_LABEL fl_define_FL_EMBOSSED_LABEL()
  Draws a label with embossed text.
• #define FL_ENGRAVED_LABEL fl_define_FL_ENGRAVED_LABEL()
  Draws a label with engraved text.
• #define FL_ICON_LABEL fl_define_FL_ICON_LABEL()
  Draws an icon as the label.
• #define FL_IMAGE_LABEL fl_define_FL_IMAGE_LABEL()
  Draws an image (Fl_Image) as the label.
• #define FL_MULTI_LABEL fl_define_FL_MULTI_LABEL()
  Draws a label that can comprise several parts like text and images.
• #define FL_SHADOW_LABEL fl_define_FL_SHADOW_LABEL()
  Draws a label with shadows behind the text.
• #define FL_SYMBOL_LABEL FL_NORMAL_LABEL
  Sets the current label type and returns its corresponding Fl_Labeltype value.

Colors

The Fl_Color type holds an FLTK color value. Colors are either 8-bit indexes into a virtual colormap or 24-bit RGB color values. (See Colors for the default FLTK colormap)

Color indices occupy the lower 8 bits of the value, while RGB colors occupy the upper 24 bits, for a byte organization of RGBI.

Fl_Color => 0xrrggbbii
  | | | +---- index between 0 and 255
  | | +------- blue color component (8 bit)
  | +-------- green component (8 bit)
  +--------- red component (8 bit)

A color can have either an index or an rgb value. Colors with rgb set and an index >0 are reserved for special use.

• const Fl_Color FL_BACKGROUND2_COLOR = 7
  the default background color for text, list, and valuactor widgets
• const Fl_Color FL_BACKGROUND_COLOR = 49
  Default background color.
• const Fl_Color FL_BLACK = 56
• const Fl_Color FL_BLUE = 216
• typedef unsigned int Fl_Color
  An FLTK color value; see also Colors

• Fl_Color fl_color_average (Fl_Color c1, Fl_Color c2, float weight)
  Returns the weighted average color between the two given colors.
• #define FL_COLOR_CUBE (Fl_Color)56
• Fl_Color fl_color_cube (int r, int g, int b)
  Returns a color out of the color cube.
• Fl_Color fl_contrast (Fl_Color fg, Fl_Color bg, Fl_Fontsize fs=0, int context=0)
  Returns a color that contrasts with the background color.
• typedef Fl_Color() Fl_Contrast_Function(Fl_Color, Fl_Color, Fl_Fontsize, int)
  Type of a custom fl_contrast() function.
• void fl_contrast_function (Fl_Contrast_Function *f)
  Register a custom contrast function.
• int fl_contrast_level ()
  Get the contrast level (sensitivity) of the fl_contrast() method.
• void fl_contrast_level (int level)
  Set the contrast level (sensitivity) of the fl_contrast() method.
• enum Fl_Contrast_Mode {
  FL_CONTRAST_NONE = 0 , FL_CONTRAST_LEGACY , FL_CONTRAST_CIELAB , FL_CONTRAST_CUSTOM ,
  FL_CONTRAST_LAST }
13.1 Enumerations.H File Reference

Define the possible modes to calculate `fl_contrast()`.

- `int fl_contrast_mode ()`
  Return the current contrast algorithm (mode).
- `void fl_contrast_mode (int mode)`
  Set the contrast algorithm (mode).

- `const Fl_Color FL_CYAN = 223`
- `const Fl_Color FL_DARK1 = 47`
- `const Fl_Color FL_DARK2 = 45`
- `const Fl_Color FL_DARK3 = 39`
- `const Fl_Color FL_DARK_BLUE = 136`
- `const Fl_Color FL_DARK_CYAN = 140`
- `const Fl_Color FL_DARK_GREEN = 60`
- `const Fl_Color FL_DARK_MAGENTA = 152`
- `const Fl_Color FL_DARK_RED = 72`
- `const Fl_Color FL_DARK_YELLOW = 76`
- `Fl_Color fl_darker (Fl_Color c)`
  Returns a darker version of the specified color.

- `const Fl_Color FL_FOREGROUND_COLOR = 0`
  the default foreground color (0) used for labels and text
- `#define FL_FREE_COLOR (Fl_Color)16`
  Colors numbered between FL_FREE_COLOR and FL_FREE_COLOR + FL_NUM_FREE_COLOR - 1 are free for the user to be given any value using Fl::set_color().

- `#define FL_GRAY (Fl_Color)32`
- `Fl_Color fl_gray_ramp (int i)`
  Returns a gray color value from black (i == 0) to white (i == FL_NUM_GRAY - 1).

- `const Fl_Color FL_GRAY0 = 32`
- `#define FL_GRAY_RAMP (Fl_Color)32`
- `Fl_Color fl_lighter (Fl_Color c)`
  Returns a lighter version of the specified color.

- `const Fl_Color FL_INACTIVE_COLOR = 8`
  the inactive foreground color

- `const Fl_Color FL_LIGHT1 = 50`
- `const Fl_Color FL_LIGHT2 = 52`
- `const Fl_Color FL_LIGHT3 = 54`
- `Fl_Color fl_lighter (Fl_Color c)`
  Returns a lighter version of the specified color.

- `double fl_lightness (Fl_Color color)`
  Return the perceived lightness of a color.
- `double fl_luminance (Fl_Color color)`
  Return the raw / physical luminance of a color.

- `const Fl_Color FL_MAGENTA = 248`
- `#define FL_NUM_BLUE 5`
- `#define FL_NUM_FREE_COLOR 16`
- `#define FL_NUM_GRAY 24`
- `#define FL_NUM_GREEN 8`
- `#define FL_NUM_RED 5`
- `const Fl_Color FL_RED = 88`
- `Fl_Color fl_rgb_color (uchar g)`
  Returns the 24-bit color value closest to g (grayscale).
- `Fl_Color fl_rgb_color (uchar r, uchar g, uchar b)`
  Returns the 24-bit color value closest to r, g, b.
- `const Fl_Color FL_SELECTION_COLOR = 15`
  the default selection/highlight color
- `const Fl_Color FL_WHITE = 255`
- `const Fl_Color FL_YELLOW = 95`
Alignment Flags

Flags to control the label alignment. This controls how the label is displayed next to or inside the widget. The default value is FL_ALIGN_CENTER (0) for most widgets, which centers the label inside the widget.

All alignment flags use the common prefix "FL_ALIGN_." In the following descriptions this prefix is sometimes omitted for brevity.

Flags can be or'd to achieve a combination of alignments, but there are some "magic values" (e.g. combinations of TOP and BOTTOM and of LEFT and RIGHT) that have special meanings (see below). For instance:

\[
\text{FL_ALIGN_TOP_LEFT} = (\text{FL_ALIGN_TOP} | \text{FL_ALIGN_LEFT}) \neq \text{FL_ALIGN_LEFT_TOP}.
\]

Outside alignments (FL_ALIGN_INSIDE is not set):

```
+---------------------------------+  
| TOP_LEFT | TOP | TOP_RIGHT |
| LEFT_TOP| | | RIGHT_TOP |
| | CENTER | | |
| LEFT | | RIGHT |
| LEFT_BOTTOM| | | RIGHT_BOTTOM |
+---------------------------------+  
| BOTTOM_LEFT | BOTTOM | BOTTOM_RIGHT |
```

Inside alignments (FL_ALIGN_INSIDE is set):

```
+---------------------------------+  
| TOP_LEFT | TOP | TOP_RIGHT |
| LEFT | CENTER | RIGHT |
| BOTTOM_LEFT | BOTTOM | BOTTOM_RIGHT |
```

See also

\( \text{Fl.Align}, \text{FL_ALIGN_CENTER}, \text{etc.} \)

Note

1. Bit positions not defined in the following constants of type \text{Fl.Align} are reserved for future extensions. Do not use.

2. The "magic values" (FL_ALIGN_LEFT, TOP_RIGHT, LEFT_BOTTOM, and RIGHT_BOTTOM) must not be used together with FL_ALIGN_INSIDE. Use TOP_LEFT, TOP_RIGHT, BOTTOM_LEFT, or BOTTOM_RIGHT instead.

3. Although bits can be or'd together there are some unused/illegal combinations, for instance:
   - setting both FL_ALIGN_TOP and FL_ALIGN_BOTTOM in combinations other than those given in the Fl.Align constants below (magic values)
   - setting both FL_ALIGN_LEFT and FL_ALIGN_RIGHT in combinations other than those given in the Fl.Align constants below (magic values)
   - using one of the "magic values" (2) together with FL_ALIGN_INSIDE

Using illegal bit combinations or undefined bits may yield unexpected behavior, and this behavior may be changed without notice in future FLTK versions.

- typedef unsigned \text{Fl.Align}

   FLTK type for alignment control.

- const \text{Fl.Align} \text{FL_ALIGN_BOTTOM} = 0x0002

   Align the label at the bottom of the widget.

- const \text{Fl.Align} \text{FL_ALIGN_BOTTOM_LEFT} = FL_ALIGN_BOTTOM | FL_ALIGN_LEFT

- const \text{Fl.Align} \text{FL_ALIGN_BOTTOM_RIGHT} = FL_ALIGN_BOTTOM | FL_ALIGN_RIGHT

- const \text{Fl.Align} \text{FL_ALIGN_CENTER} = 0x0000

   Align the label horizontally in the middle.

- const \text{Fl.Align} \text{FL_ALIGN_CLIP} = 0x0040

   All parts of the label that are larger than the widget will not be drawn.

- const \text{Fl.Align} \text{FL_ALIGN_IMAGE_BACKDROP} = 0x0200

   If the label contains an image, draw the image or deimage in the background.
• const Fl_Align FL_ALIGN_IMAGE_MASK = 0x0320
  Mask value to test for image alignment flags.
• const Fl_Align FL_ALIGN_IMAGE_NEXT_TO_TEXT = 0x0100
  If the label contains an image, draw the text to the right of the image.
• const Fl_Align FL_ALIGN_IMAGE_OVER_TEXT = 0x0000
  If the label contains an image, draw the text below the image.
• const Fl_Align FL_ALIGN_INSIDE = 0x0010
  Draw the label inside of the widget.
• const Fl_Align FL_ALIGN_LEFT = 0x0004
  Align the label at the left of the widget.
• const Fl_Align FL_ALIGN_LEFT_BOTTOM = 0x000d
  Outside only, left of widget, bottom position, magic value: TOP | LEFT | RIGHT.
• const Fl_Align FL_ALIGN_LEFT_TOP = 0x0007
  Outside only, left of widget, top position, magic value: TOP | BOTTOM | LEFT.
• const Fl_Align FL_ALIGN_NOWRAP = 0x0000
  Nothing, same as FL_ALIGN_CENTER, for back compatibility.
• const Fl_Align FL_ALIGN_POSITION_MASK = 0x000f
  Mask value to test for TOP, BOTTOM, LEFT, and RIGHT flags.
• const Fl_Align FL_ALIGN_RIGHT = 0x0008
  Align the label to the right of the widget.
• const Fl_Align FL_ALIGN_RIGHT_BOTTOM = 0x000e
  Outside only, right of widget, bottom position, magic value: BOTTOM | LEFT | RIGHT.
• const Fl_Align FL_ALIGN_RIGHT_TOP = 0x000b
  Outside only, right of widget, top position, magic value: TOP | BOTTOM | RIGHT.
• const Fl_Align FL_ALIGN_TEXT_NEXT_TO_IMAGE = 0x0120
  If the label contains an image, draw the text to the left of the image.
• const Fl_Align FL_ALIGN_TEXT_OVER_IMAGE = 0x0020
  If the label contains an image, draw the text on top of the image.
• const Fl_Align FL_ALIGN_TOP = 0x0001
  Align the label at the top of the widget.
• const Fl_Align FL_ALIGN_TOP_LEFT = FL_ALIGN_TOP | FL_ALIGN_LEFT
• const Fl_Align FL_ALIGN_TOP_RIGHT = FL_ALIGN_TOP | FL_ALIGN_RIGHT
• const Fl_Align FL_ALIGN_WRAP = 0x0080
  Wrap text that does not fit the width of the widget.

Font Numbers
The following constants define the standard FLTK fonts:

• const Fl_Font FL_BOLD = 1
  add this to Helvetica, Courier, or Times
• const Fl_Font FL_BOLD_ITALIC = 3
  add this to Helvetica, Courier, or Times
• const Fl_Font FL_COURIER = 4
  Courier normal.
• const Fl_Font FL_COURIER_BOLD = 5
  Courier bold.
• const Fl_Font FL_COURIER_BOLD_ITALIC = 7
  Courier bold-italic.
• const Fl_Font FL_COURIER_ITALIC = 6
  Courier italic.
• typedef int Fl_Font

Generated by Doxygen
A font number is an index into the internal font table.

- const Fl_Font FL_FREE_FONT = 16
  first one to allocate
- const Fl_Font FL_HELVETICA = 0
  Helvetica (or Arial) normal (0)
- const Fl_Font FL_HELVETICA_BOLD = 1
  Helvetica (or Arial) bold.
- const Fl_Font FL_HELVETICA_BOLD_ITALIC = 3
  Helvetica (or Arial) bold-oblique.
- const Fl_Font FL_HELVETICA_ITALIC = 2
  Helvetica (or Arial) oblique.
- const Fl_Font FLITALIC = 2
  add this to Helvetica, Courier, or Times
- const Fl_Font FL_SCREEN = 13
  Default monospaced screen font.
- const Fl_Font FL_SCREEN_BOLD = 14
  Default monospaced bold screen font.
- const Fl_Font FL_SYMBOL = 12
  Standard symbol font.
- const Fl_Font FL_TIMES = 8
  Times roman.
- const Fl_Font FL_TIMES_BOLD = 9
  Times roman bold.
- const Fl_Font FL_TIMES_BOLD_ITALIC = 11
  Times roman bold-italic.
- const Fl_Font FL_TIMES_ITALIC = 10
  Times roman italic.
- const Fl_Font FL_ZAPF_DINGBATS = 15
  Zapf-dingbats font.

13.1.1 Detailed Description
This file contains type definitions and general enumerations.

13.1.2 Macro Definition Documentation

13.1.2.1 FL_ABI_VERSION

#define FL_ABI_VERSION FL_API_VERSION

The FLTK ABI (Application Binary Interface) version number as an int.
FL_ABI_VERSION is an int that describes the major, minor, and patch ABI version numbers in the same format as FL_API_VERSION.
The ABI version number FL_ABI_VERSION is usually the same as the API version FL_API_VERSION with the last two digits set to '00'.
FLTK retains the ABI (Application Binary Interface) during patch releases of the same major and minor versions. Examples:

<table>
<thead>
<tr>
<th>Version</th>
<th>FL_API_VERSION</th>
<th>FL_ABI_VERSION</th>
<th>FL_VERSION (deprecated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.0</td>
<td>10300</td>
<td>10300</td>
<td>1.0300</td>
</tr>
<tr>
<td>1.3.4</td>
<td>10304</td>
<td>10300</td>
<td>1.0304</td>
</tr>
</tbody>
</table>

Version 1.2.3 is actually stored as 10203 to allow for more than 9 minor and patch releases.
The FL_MAJOR_VERSION, FL_MINOR_VERSION, and FL_PATCH_VERSION constants give the integral values for the major, minor, and patch releases respectively.
To enable new ABI-breaking features in patch releases you can configure FLTK to use a higher FL_ABI_VERSION.
See also

README.abi-version.txt

### 13.1.2.2 FL_API_VERSION

```c
#define FL_API_VERSION (FL_MAJOR_VERSION*10000 + FL_MINOR_VERSION*100 + FL_PATCH_VERSION)
```

The FLTK API version number as an `int`.

FL_API_VERSION is an `int` that describes the major, minor, and patch version numbers.

Version 1.2.3 is actually stored as 10203 to allow for more than 9 minor and patch releases.

The FL_MAJOR_VERSION, FL_MINOR_VERSION, and FL_PATCH_VERSION constants give the integral values for the major, minor, and patch releases respectively.

**Note**

FL_API_VERSION is intended to replace the deprecated `double` FL_VERSION.

See also

Fl::api_version()

### 13.1.2.3 FL_IMAGE_LABEL

```c
#define FL_IMAGE_LABEL fl_define_FL_IMAGE_LABEL()
```

Draws an image (`Fl::Image`) as the label.

This is useful for one particular part of an `Fl::Multi_Label`. Use `Fl_Widget::image()` and/or `Fl_Widget::deimage()` for normal widgets with images as labels.

### 13.1.2.4 FL_MAJOR_VERSION

```c
#define FL_MAJOR_VERSION 1
```

The major release version of this FLTK library.

See also

FL_VERSION

### 13.1.2.5 FL_MINOR_VERSION

```c
#define FL_MINOR_VERSION 4
```

The minor release version for this library.

FLTK remains mostly source-code compatible between minor version changes.

### 13.1.2.6 FL_MULTI_LABEL

```c
#define FL_MULTI_LABEL fl_define_FL_MULTI_LABEL()
```

Draws a label that can comprise several parts like text and images.

See also

Fl::Multi_Label

### 13.1.2.7 FL_PATCH_VERSION

```c
#define FL_PATCH_VERSION 0
```

The patch version for this library.

FLTK remains binary compatible between patches.
13.1.2.8 FL_SYMBOL_LABEL

#define FL_SYMBOL_LABEL FL_NORMAL_LABEL
Sets the current label type and returns its corresponding Fl_Labeltype value.
FL_SYMBOL_LABEL is an alias for FL_NORMAL_LABEL.
'@' symbols can be drawn with normal labels as well.
This definition is for historical reasons only (forms compatibility). You should use FL_NORMAL_LABEL instead.

13.1.2.9 FL_VERSION

#define FL_VERSION
Value:

    ( (double)FL_MAJOR_VERSION + \\
      (double)FL_MINOR_VERSION * 0.01 + \\
      (double)FL_PATCH_VERSION * 0.0001 )

The FLTK version number as a double.
FL_VERSION is a double that describes the major, minor, and patch version numbers.
Version 1.2.3 is actually stored as 1.0203 to allow for more than 9 minor and patch releases.

Deprecated This double version number is retained for compatibility with existing program code. New code
should use int FL_API_VERSION instead. FL_VERSION is deprecated because comparisons of
floating point values may fail due to rounding errors. However, there are currently no plans to remove
this deprecated constant.

FL_VERSION is equivalent to (double)FL_API_VERSION / 10000.

See also

Fl::version() (deprecated as well)
FL_API_VERSION
Fl::api_version()

13.1.3 Typedef Documentation

13.1.3.1 Fl_Contrast_Function
typedef Fl_Color() Fl_Contrast_Function(Fl_Color, Fl_Color, Fl_Fontsize, int)
Type of a custom fl_contrast() function.
Use this signature to define your own custom fl_contrast() function together with fl_contrast_mode(FL_CONTRAST_CUSTOM). Example:

Fl_Color my_contrast(Fl_Color fg, Fl_Color bg, Fl_Fontsize fs, int context) {
    // calculate contrast and ...
    return color;
}

// call this early in your main() program:
fl_contrast_function(my_contrast);
fl_contrast_mode(FL_CONTRAST_CUSTOM);

See also

fl_contrast(Fl_Color, Fl_Color, Fl_Fontsize, int)
fl_contrast_mode(int)

13.1.3.2 Fl_Fontsize
typedef int Fl_Fontsize
Size of a font in pixels.
This is the approximate height of a font in pixels.

13.1.4 Enumeration Type Documentation

13.1.4.1 anonymous enum
anonymous enum
FD "when" conditions.
13.1 Enumerations.H File Reference

### 13.1.4.2 Fl_Arrow_Type

enum Fl_Arrow_Type

Arrow types define the type of arrow drawing function.

FLTK schemes can draw several graphical elements in their particular way. One of these elements is an arrow type that can be in different GUI elements like scrollbars, choice buttons, and FLTK's `Fl_Return_Button`.

**Note**

This enum is not yet stable (as of FLTK 1.4.0) and may be changed without notice as necessary.

**Since**

1.4.0

**Enumerator**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_ARROW_SINGLE</td>
<td>Single arrow, e.g. in <code>Fl_Scrollbar</code>.</td>
</tr>
<tr>
<td>FL_ARROW_DOUBLE</td>
<td>Double arrow, e.g. in <code>Fl_Counter</code>.</td>
</tr>
<tr>
<td>FL_ARROW_CHOICE</td>
<td>Dropdown box, e.g. in <code>Fl_Choice</code>.</td>
</tr>
<tr>
<td>FL_ARROW_RETURN</td>
<td>Return arrow, e.g. in <code>Fl_Return_Button</code>.</td>
</tr>
</tbody>
</table>

### 13.1.4.3 Fl_Boxtype

enum Fl_Boxtype

FLTK standard box types.

This enum defines the standard box types included with FLTK.

**Note**

The documented enum Fl_Boxtype contains some values (names) with leading underscores, e.g. `FL_SHADOW_BOX`. This is due to technical reasons - please use the same values (names) without the leading underscore in your code! Enum values with leading underscores are reserved for internal use and subject to change without notice!

**Enumerator**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_NO_BOX</td>
<td>nothing is drawn at all, this box is invisible</td>
</tr>
<tr>
<td>FL_FLAT_BOX</td>
<td>a flat box</td>
</tr>
<tr>
<td>FL_UP_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_DOWN_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_UP_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_DOWN_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_THIN_UP_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_THIN_DOWN_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_THIN_UP_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_THIN_DOWN_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_ENGRAVED_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
</tbody>
</table>

Generated by Doxygen
<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_EMBOSSED_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_ENGRAVED_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_EMBOSSED_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_BORDER_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>_FL_SHADOW_BOX</td>
<td>see figure Standard Box Types, use FL_SHADOW_BOX</td>
</tr>
<tr>
<td>FL_SHADOW_FRAME</td>
<td>see figure Standard Box Types, use FL_SHADOW_FRAME</td>
</tr>
<tr>
<td>_FL_ROUNDED_BOX</td>
<td>see figure Standard Box Types, use FL_ROUNDED_BOX</td>
</tr>
<tr>
<td>FL_RSHADOW_BOX</td>
<td>see figure Standard Box Types, use FL_RSHADOW_BOX</td>
</tr>
<tr>
<td>_FL_ROUNDED_FRAME</td>
<td>see figure Standard Box Types, use FL_ROUNDED_FRAME</td>
</tr>
<tr>
<td>_FL_RFLAT_BOX</td>
<td>see figure Standard Box Types, use FL_RFLAT_BOX</td>
</tr>
<tr>
<td>_FL_SHADOW_BOX</td>
<td>see figure Standard Box Types, use FL_SHADOW_BOX</td>
</tr>
<tr>
<td>FL_SHADOW_FRAME</td>
<td>see figure Standard Box Types, use FL_SHADOW_FRAME</td>
</tr>
<tr>
<td>_FL_ROUND_UP_BOX</td>
<td>see figure Standard Box Types, use FL_ROUND_UP_BOX</td>
</tr>
<tr>
<td>_FL_ROUND_DOWN_BOX</td>
<td>see figure Standard Box Types, use FL_ROUND_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_DIAMOND_UP_BOX</td>
<td>see figure Standard Box Types, use FL_DIAMOND_UP_BOX</td>
</tr>
<tr>
<td>_FL_DIAMOND_DOWN_BOX</td>
<td>see figure Standard Box Types, use FL_DIAMOND_DOWN_BOX</td>
</tr>
<tr>
<td>FL_OVAL_BOX</td>
<td>see figure Standard Box Types, use FL_OVAL_BOX</td>
</tr>
<tr>
<td>FL_OSHADOW_BOX</td>
<td>see figure Standard Box Types, use FL_OSHADOW_BOX</td>
</tr>
<tr>
<td>FL_OVAL_FRAME</td>
<td>see figure Standard Box Types, use FL_OVAL_FRAME</td>
</tr>
<tr>
<td>_FL_OFLAT_BOX</td>
<td>see figure Standard Box Types, use FL_OFLAT_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_UP_BOX</td>
<td>plastic version of FL_UP_BOX, use FL_PLASTIC_UP_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_DOWN_BOX</td>
<td>plastic version of FL_DOWN_BOX, use FL_PLASTIC_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_UP_FRAME</td>
<td>plastic version of FL_UP_FRAME, use FL_PLASTIC_UP_FRAME</td>
</tr>
<tr>
<td>_FL_PLASTIC_DOWN_FRAME</td>
<td>plastic version of FL_DOWN_FRAME, use FL_PLASTIC_DOWN_FRAME</td>
</tr>
<tr>
<td>_FL_PLASTIC_THIN_UP_BOX</td>
<td>plastic version of FL_THIN_UP_BOX, use FL_PLASTIC_THIN_UP_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_THIN_DOWN_BOX</td>
<td>plastic version of FL_THIN_DOWN_BOX, use FL_PLASTIC_THIN_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_ROUND_UP_BOX</td>
<td>plastic version of FL_ROUND_UP_BOX, use FL_PLASTIC_ROUND_UP_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_ROUND_DOWN_BOX</td>
<td>plastic version of FL_ROUND_DOWN_BOX, use FL_PLASTIC_ROUND_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_GTK_UP_BOX</td>
<td>gtk+ version of FL_UP_BOX, use FL_GTK_UP_BOX</td>
</tr>
<tr>
<td>_FL_GTK_DOWN_BOX</td>
<td>gtk+ version of FL_DOWN_BOX, use FL_GTK_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_GTK_UP_FRAME</td>
<td>gtk+ version of FL_UP_FRAME, use FL_GTK_UP_FRAME</td>
</tr>
<tr>
<td>_FL_GTK_DOWN_FRAME</td>
<td>gtk+ version of FL_DOWN_FRAME, use FL_GTK_DOWN_FRAME</td>
</tr>
<tr>
<td>_FL_GTK_THIN_UP_BOX</td>
<td>gtk+ version of FL_THIN_UP_BOX, use FL_GTK_THIN_UP_BOX</td>
</tr>
<tr>
<td>_FL_GTK_THIN_DOWN_BOX</td>
<td>gtk+ version of FL_THIN_DOWN_BOX, use FL_GTK_THIN_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_GTK_THIN_UP_FRAME</td>
<td>gtk+ version of FL_THIN_UP_FRAME, use FL_GTK_THIN_UP_FRAME</td>
</tr>
<tr>
<td>_FL_GTK_THIN_DOWN_FRAME</td>
<td>gtk+ version of FL_THIN_DOWN_FRAME, use FL_GTK_THIN_DOWN_FRAME</td>
</tr>
<tr>
<td>_FL_GTK_ROUND_UP_BOX</td>
<td>gtk+ version of FL_ROUND_UP_BOX, use FL_GTK_ROUND_UP_BOX</td>
</tr>
<tr>
<td>_FL_GTK_ROUND_DOWN_BOX</td>
<td>gtk+ version of FL_ROUND_DOWN_BOX, use FL_GTK_ROUND_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_GLEAM_UP_BOX</td>
<td>gleam version of FL_UP_BOX, use FL_GLEAM_UP_BOX</td>
</tr>
<tr>
<td>_FL_GLEAM_DOWN_BOX</td>
<td>gleam version of FL_DOWN_BOX, use FL_GLEAM_DOWN_BOX</td>
</tr>
</tbody>
</table>
13.1 Enumerations.H File Reference

### Enumerator

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_FL_GLEAM_UP_FRAME</td>
<td>gleam version of FL_UP_FRAME, use FL_GLEAM_UP_FRAME</td>
</tr>
<tr>
<td>_FL_GLEAM_DOWN_FRAME</td>
<td>gleam version of FL_DOWN_FRAME, use FL_GLEAM_DOWN_FRAME</td>
</tr>
<tr>
<td>_FL_GLEAM_THIN_UP_BOX</td>
<td>gleam version of FL_THIN_UP_BOX, use FL_GLEAM_THIN_UP_BOX</td>
</tr>
<tr>
<td>_FL_GLEAM_THIN_DOWN_BOX</td>
<td>gleam version of FL_THIN_DOWN_BOX, use FL_GLEAM_THIN_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_GLEAM_ROUND_UP_BOX</td>
<td>gleam version of FL_ROUND_UP_BOX, use FL_GLEAM_ROUND_UP_BOX</td>
</tr>
<tr>
<td>_FL_GLEAM_ROUND_DOWN_BOX</td>
<td>gleam version of FL_ROUND_DOWN_BOX, use FL_GLEAM_ROUND_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_OXY_UP_BOX</td>
<td>oxy version of FL_UP_BOX, use FL_OXY_UP_BOX</td>
</tr>
<tr>
<td>_FL_OXY_DOWN_BOX</td>
<td>oxy version of FL_DOWN_BOX, use FL_OXY_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_OXY_UP_FRAME</td>
<td>oxy version of FL_UP_FRAME, use FL_OXY_UP_FRAME</td>
</tr>
<tr>
<td>_FL_OXY_DOWN_FRAME</td>
<td>oxy version of FL_DOWN_FRAME, use FL_OXY_DOWN_FRAME</td>
</tr>
<tr>
<td>_FL_OXY_THIN_UP_BOX</td>
<td>oxy version of FL_THIN_UP_BOX, use FL_OXY_THIN_UP_BOX</td>
</tr>
<tr>
<td>_FL_OXY_THIN_DOWN_BOX</td>
<td>oxy version of FL_THIN_DOWN_BOX, use FL_OXY_THIN_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_OXY_ROUND_UP_BOX</td>
<td>oxy version of FL_ROUND_UP_BOX, use FL_OXY_ROUND_UP_BOX</td>
</tr>
<tr>
<td>_FL_OXY_ROUND_DOWN_BOX</td>
<td>oxy version of FL_ROUND_DOWN_BOX, use FL_OXY_ROUND_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_OXY_BUTTON_UP_BOX</td>
<td>FL_OXY_BUTTON_UP_BOX (new boxtype ?), use FL_OXY_BUTTON_UP_BOX</td>
</tr>
<tr>
<td>_FL_OXY_BUTTON_DOWN_BOX</td>
<td>FL_OXY_BUTTON_DOWN_BOX (new boxtype ?), use FL_OXY_BUTTON_DOWN_BOX</td>
</tr>
<tr>
<td>FL_FREE_BOXTYPE</td>
<td>the first free box type for creation of new box types</td>
</tr>
</tbody>
</table>

### 13.1.4.4 Fl_Callback_Reason

enum Fl_Callback_Reason

These constants describe why a callback is performed.

See also

Fl::callback_reason(), Fl_Widget::when(), Fl_When

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_REASON_UNKNOWN</td>
<td>unknown or unset reason</td>
</tr>
<tr>
<td>FL_REASON_SELECTED</td>
<td>an item was selected</td>
</tr>
<tr>
<td>FL_REASON_DESELECTED</td>
<td>an item was de-selected</td>
</tr>
<tr>
<td>FL_REASON_RESELECTED</td>
<td>an item was re-selected (double-clicked).</td>
</tr>
<tr>
<td>FL_REASON_OPENED</td>
<td>an item was opened</td>
</tr>
<tr>
<td>FL_REASON_CLOSED</td>
<td>an item was closed</td>
</tr>
<tr>
<td>FL_REASON_DRAGGED</td>
<td>an item was dragged into a new place</td>
</tr>
<tr>
<td>FL_REASON_CANCELLED</td>
<td>a dialog was cancelled</td>
</tr>
</tbody>
</table>
13.1.4.5 Fl_Contrast_Mode

```cpp
enum Fl_Contrast_Mode
Define the possible modes to calculate fl_contrast().
```

#### Enumerator

<table>
<thead>
<tr>
<th>Enumerators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_CONTRAST_NONE</td>
<td>always return foreground color</td>
</tr>
<tr>
<td>FL_CONTRAST_LEGACY</td>
<td>legacy (FLTK 1.3.x) contrast function</td>
</tr>
<tr>
<td>FL_CONTRAST_CIELAB</td>
<td>new (FLTK 1.4.0) default function</td>
</tr>
<tr>
<td>FL_CONTRAST_CUSTOM</td>
<td>optional custom contrast function</td>
</tr>
<tr>
<td>FL_CONTRAST_LAST</td>
<td>internal use only (invalid contrast mode)</td>
</tr>
</tbody>
</table>

13.1.4.6 Fl_Cursor

```cpp
enum Fl_Cursor
The following constants define the mouse cursors that are available in FLTK. Cursors are provided by the system when available, or bitmaps built into FLTK as a fallback.
```

#### Enumerator

<table>
<thead>
<tr>
<th>Enumerators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_CURSOR_DEFAULT</td>
<td>the default cursor, usually an arrow:</td>
</tr>
<tr>
<td>FL_CURSOR_ARROW</td>
<td>an arrow pointer:</td>
</tr>
<tr>
<td>FL_CURSOR_CROSS</td>
<td>crosshair:</td>
</tr>
<tr>
<td>FL_CURSOR_WAIT</td>
<td>busy indicator (for instance hourglass): ,</td>
</tr>
<tr>
<td>FL_CURSOR_INSERT</td>
<td>I-beam:</td>
</tr>
<tr>
<td>FL_CURSOR_HAND</td>
<td>pointing hand:</td>
</tr>
<tr>
<td>FL_CURSOR_HELP</td>
<td>question mark pointer: ?</td>
</tr>
<tr>
<td>FL_CURSOR_MOVE</td>
<td>4-pointed arrow or hand: ,</td>
</tr>
<tr>
<td>FL_CURSOR_NS</td>
<td>up/down resize:</td>
</tr>
<tr>
<td>FL_CURSOR_WE</td>
<td>left/right resize:</td>
</tr>
<tr>
<td>FL_CURSOR_NWSE</td>
<td>diagonal resize:</td>
</tr>
<tr>
<td>FL_CURSOR_NESW</td>
<td>diagonal resize:</td>
</tr>
<tr>
<td>FL_CURSOR_N</td>
<td>upwards resize:</td>
</tr>
<tr>
<td>FL_CURSOR_NE</td>
<td>upwards, right resize:</td>
</tr>
<tr>
<td>FL_CURSOR_E</td>
<td>rightwards resize:</td>
</tr>
<tr>
<td>FL_CURSOR_SE</td>
<td>downwards, right resize:</td>
</tr>
<tr>
<td>FL_CURSOR_S</td>
<td>downwards resize:</td>
</tr>
<tr>
<td>FL_CURSOR_SW</td>
<td>downwards, left resize:</td>
</tr>
<tr>
<td>FL_CURSOR_W</td>
<td>leftwards resize:</td>
</tr>
<tr>
<td>FL_CURSOR_NW</td>
<td>upwards, left resize:</td>
</tr>
<tr>
<td>FL_CURSOR_NONE</td>
<td>invisible.</td>
</tr>
</tbody>
</table>
13.1.4.7 Fl_Damage

enum Fl_Damage
Damage masks.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_DAMAGE_CHILD</td>
<td>A child needs to be redrawn.</td>
</tr>
<tr>
<td>FL_DAMAGE_EXPOSE</td>
<td>The window was exposed.</td>
</tr>
<tr>
<td>FL_DAMAGE_SCROLL</td>
<td>The Fl_Scroll widget was scrolled. Used by other widgets for other widget specific damages.</td>
</tr>
<tr>
<td>FL_DAMAGE_OVERLAY</td>
<td>The overlay planes need to be redrawn.</td>
</tr>
<tr>
<td>FL_DAMAGE_USER1</td>
<td>First user-defined damage bit.</td>
</tr>
<tr>
<td>FL_DAMAGE_USER2</td>
<td>Second user-defined damage bit.</td>
</tr>
<tr>
<td>FL_DAMAGE_ALL</td>
<td>Everything needs to be redrawn.</td>
</tr>
</tbody>
</table>

13.1.4.8 Fl_Event

enum Fl_Event
Every time a user moves the mouse pointer, clicks a button, or presses a key, an event is generated and sent to your application.

Events can also come from other programs like the window manager.

Events are identified by the integer argument passed to the Fl_Widget::handle() virtual method. Other information about the most recent event is stored in static locations and acquired by calling the Fl::event_*() methods. This static information remains valid until the next event is read from the window system, so it is ok to look at it outside of the handle() method.

Event numbers can be converted to their actual names using the fl_eventnames[] array defined in #include <FL/names.h>

See also
Fl::event_text(), Fl::event_key(), class Fl::

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_NO_EVENT</td>
<td>No event.</td>
</tr>
<tr>
<td>FL_PUSH</td>
<td>A mouse button has gone down with the mouse pointing at this widget. You can find out what button by calling Fl::event_button(). You find out the mouse position by calling Fl::event_x() and Fl::event_y(). A widget indicates that it &quot;wants&quot; the mouse click by returning non-zero from its Fl_Widget::handle() method. It will then become the Fl::pushed() widget and will get FL_DRAG and the matching FL_RELEASE events. If Fl_Widget::handle() returns zero then FLTK will try sending the FL_PUSH to another widget.</td>
</tr>
<tr>
<td>FL_RELEASE</td>
<td>A mouse button has been released. You can find out what button by calling Fl::event_button(). In order to receive the FL_RELEASE event, the widget must return non-zero when handling FL_PUSH.</td>
</tr>
<tr>
<td>FL_ENTER</td>
<td>The mouse has been moved to point at this widget. This can be used for highlighting feedback. If a widget wants to highlight or otherwise track the mouse, it indicates this by returning non-zero from its handle() method. It then becomes the Fl::belowmouse() widget and will receive FL_MOVE and FL_LEAVE events.</td>
</tr>
</tbody>
</table>
## Enumerators

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_LEAVE</td>
<td>The mouse has moved out of the widget. In order to receive the FL_LEAVE event, the widget must return non-zero when handling FL_ENTER.</td>
</tr>
<tr>
<td>FL_DRAG</td>
<td>The mouse has moved with a button held down. The current button state is in Fl::event_state(). The mouse position is in Fl::event_x() and Fl::event_y(). In order to receive FL_DRAG events, the widget must return non-zero when handling FL_PUSH.</td>
</tr>
<tr>
<td>FL_FOCUS</td>
<td>This indicates an attempt to give a widget the keyboard focus. If a widget wants the focus, it should change itself to display the fact that it has the focus, and return non-zero from its handle() method. It then becomes the Fl::focus() widget and gets FL_KEYDOWN, FL_KEYUP, and FL_UNFOCUS events. The focus will change either because the window manager changed which window gets the focus, or because the user tried to navigate using tab, arrows, or other keys. You can check Fl::event_key() to figure out why it moved. For navigation it will be the key pressed and for interaction with the window manager it will be zero.</td>
</tr>
<tr>
<td>FL_UNFOCUS</td>
<td>This event is sent to the previous Fl::focus() widget when another widget gets the focus or the window loses focus.</td>
</tr>
<tr>
<td>FL_KEYDOWN</td>
<td>A key was pressed (FL_KEYDOWN) or released (FL_KEYUP). FL_KEYBOARD is a synonym for FL_KEYDOWN. The key can be found in Fl::event_key(). The text that the key should insert can be found with Fl::event_text() and its length is in Fl::event_length(). If you use the key handle() should return 1. If you return zero then FLTK assumes you ignored the key and will then attempt to send it to a parent widget. If none of them want it, it will change the event into a FL_SHORTCUT event. To receive FL_KEYBOARD events you must also respond to the FL_FOCUS and FL_UNFOCUS events. If you are writing a text-editing widget you may also want to call the Fl::compose() function to translate individual keystrokes into non-ASCII characters. FL_KEYUP events are sent to the widget that currently has focus. This is not necessarily the same widget that received the corresponding FL_KEYDOWN event because focus may have changed between events.</td>
</tr>
<tr>
<td>FL_KEYBOARD</td>
<td>Equivalent to FL_KEYDOWN. See also FL_KEYDOWN</td>
</tr>
<tr>
<td>FL_KEYUP</td>
<td>Key release event. See also FL_KEYDOWN</td>
</tr>
<tr>
<td>FL_CLOSE</td>
<td>The user clicked the close button of a window. This event is used internally only to trigger the callback of Fl_Window derived classes. The default callback closes the window calling Fl_Window::hide().</td>
</tr>
<tr>
<td>Enumerator</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FL_MOVE</td>
<td>The mouse has moved without any mouse buttons held down. This event is sent to the <code>Fl::belowmouse()</code> widget. In order to receive FL_MOVE events, the widget must return non-zero when handling FL_ENTER.</td>
</tr>
<tr>
<td>FL_SHORTCUT</td>
<td>If the <code>Fl::focus()</code> widget is zero or ignores an FL_KEYBOARD event then FLTK tries sending this event to every widget it can, until one of them returns non-zero. FL_SHORTCUT is first sent to the <code>Fl::belowmouse()</code> widget, then its parents and siblings, and eventually to every widget in the window, trying to find an object that returns non-zero. FLTK tries really hard to not to ignore any keystrokes! You can also make &quot;global&quot; shortcuts by using <code>Fl::add_handler()</code>. A global shortcut will work no matter what windows are displayed or which one has the focus.</td>
</tr>
<tr>
<td>FL_DEACTIVATE</td>
<td>This widget is no longer active, due to <code>Fl_Widget::deactivate()</code> being called on it or one of its parents. <code>Fl_Widget::active()</code> may still be true after this, the widget is only active if <code>Fl_Widget::active()</code> is true on it and all its parents (use <code>Fl_Widget::active_r()</code> to check this).</td>
</tr>
<tr>
<td>FL_ACTIVATE</td>
<td>This widget is now active, due to <code>Fl_Widget::activate()</code> being called on it or one of its parents.</td>
</tr>
<tr>
<td>FL_HIDE</td>
<td>This widget is no longer visible, due to <code>Fl_Widget::hide()</code> being called on it or one of its parents, or due to a parent window being minimized. <code>Fl_Widget::visible()</code> may still be true after this, but the widget is visible only if visible() is true for it and all its parents (use <code>Fl_Widget::visible_r()</code> to check this).</td>
</tr>
<tr>
<td>FL_SHOW</td>
<td>This widget is visible again, due to <code>Fl_Widget::show()</code> being called on it or one of its parents, or due to a parent window being restored. Child <code>Fl_Windows</code> respond to this by actually creating the window if not done already, so if you subclass a window, be sure to pass FL_SHOW to the base class <code>Fl_Widget::handle()</code> method!</td>
</tr>
<tr>
<td>FL_PASTE</td>
<td>You should get this event some time after you call <code>Fl::paste()</code>. The contents of <code>Fl::event_text()</code> is the text to insert and the number of characters is in <code>Fl::event_length()</code>.</td>
</tr>
<tr>
<td>FL_SELECTIONCLEAR</td>
<td>The <code>Fl::selection_owner()</code> will get this event before the selection is moved to another widget. This indicates that some other widget or program has claimed the selection. Motif programs used this to clear the selection indication. Most modern programs ignore this.</td>
</tr>
<tr>
<td>FL_MOUSEWHEEL</td>
<td>The user has moved the mouse wheel. The <code>Fl::event_dx()</code> and <code>Fl::event_dy()</code> methods can be used to find the amount to scroll horizontally and vertically.</td>
</tr>
<tr>
<td>FL_DND_ENTER</td>
<td>The mouse has been moved to point at this widget. A widget that is interested in receiving drag'n'drop data must return 1 to receive FL_DND_DRAG, FL_DND_LEAVE and FL_DND_RELEASE events.</td>
</tr>
<tr>
<td>FL_DND_DRAG</td>
<td>The mouse has been moved inside a widget while dragging data. A widget that is interested in receiving drag'n'drop data should indicate the possible drop position.</td>
</tr>
<tr>
<td>FL_DND_LEAVE</td>
<td>The mouse has moved out of the widget.</td>
</tr>
<tr>
<td>FL_DND_RELEASE</td>
<td>The user has released the mouse button dropping data into the widget. If the widget returns 1, it will receive the data in the immediately following FL_PASTE event.</td>
</tr>
</tbody>
</table>
## 13.1.4.9 Fl_Labeltype

**enum Fl_Labeltype**

The labeltype() method sets the type of the label.

**Note**

The documented enum Fl_Labeltype contains some values (names) with leading underscores, e.g. _FL_IMAGE_LABEL. This is due to technical reasons - please use the same values (names) without the leading underscore in your code! Enum values with leading underscores are reserved for internal use and subject to change without notice!

The following standard label types are included:

**Enumerator**

<table>
<thead>
<tr>
<th>Labeltype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_NORMAL_LABEL</td>
<td>draws the text (0)</td>
</tr>
<tr>
<td>FL_NO_LABEL</td>
<td>does nothing</td>
</tr>
<tr>
<td>_FL_SHADOW_LABEL</td>
<td>draws a drop shadow under the text</td>
</tr>
<tr>
<td>_FL_ENGRAVED_LABEL</td>
<td>draws edges as thought the text is engraved</td>
</tr>
<tr>
<td>_FL_EMBOSSED_LABEL</td>
<td>draws edges as thought the text is raised</td>
</tr>
<tr>
<td>_FL_MULTI_LABEL</td>
<td>draws a composite label</td>
</tr>
<tr>
<td>_FL_ICON_LABEL</td>
<td>draws the icon associated with the text</td>
</tr>
<tr>
<td>_FL_IMAGE_LABEL</td>
<td>the label displays an &quot;icon&quot; based on a Fl_Image</td>
</tr>
<tr>
<td>FL_FREE_LABELTYPE</td>
<td>first free labeltype to use for creating own labeltypes</td>
</tr>
</tbody>
</table>

## 13.1.4.10 Fl_Orientation

**enum Fl_Orientation**

Fl_Orientation describes the orientation of a GUI element.

FLTK schemes can draw several graphical elements, for instance arrows, pointing at different directions. This enum defines the direction to use for drawing a particular GUI element.

The definition of this enum was chosen such that the enum value can be multiplied by 45 to get a rotation angle in degrees starting at the horizontal axis (0 = right, 1 = NE, 2 = up, ...) that can be used with fl_rotate(). Note: angle is counter-clockwise in degrees.

The 'unspecified' value FL_ORIENTATION_NONE shall be used for elements that would usually not be rotated, like the return arrow of the Fl_Return_Button. It can still be used as an angle though since it is the same value as FL_Orient_Right (0 degrees).
This enum is not yet stable (as of FLTK 1.4.0) and may be changed without notice as necessary.

Since
1.4.0

Enumerator

<table>
<thead>
<tr>
<th>FL_ORIENTATION</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_ORIENTATION_NONE</td>
<td>GUI element direction is unspecified.</td>
</tr>
<tr>
<td>FL_ORIENTATION_RIGHT</td>
<td>GUI element pointing right (0°)</td>
</tr>
<tr>
<td>FL_ORIENTATION_NE</td>
<td>GUI element pointing NE (45°)</td>
</tr>
<tr>
<td>FL_ORIENTATION_UP</td>
<td>GUI element pointing up (90°)</td>
</tr>
<tr>
<td>FL_ORIENTATION_NW</td>
<td>GUI element pointing NW (135°)</td>
</tr>
<tr>
<td>FL_ORIENTATION_LEFT</td>
<td>GUI element pointing left (180°)</td>
</tr>
<tr>
<td>FL_ORIENTATION_SW</td>
<td>GUI element pointing SW (225°)</td>
</tr>
<tr>
<td>FL_ORIENTATION_DOWN</td>
<td>GUI element pointing down (270°)</td>
</tr>
<tr>
<td>FL_ORIENTATION_SE</td>
<td>GUI element pointing SE (315°)</td>
</tr>
</tbody>
</table>

13.1.4.11 Fl_When

enum Fl_When

These constants determine when a callback is performed. Fl_When is a bit field. Some values are merely shortcuts for common bit combinations. New flags may be added in the future, so it's important to mask the required bit when reading via when().

Note

Some widgets may not fully support Fl_WHEN_... flags.

See also

Fl_Widget::when(), Fl::callback_reason(), Fl_Callback_Reason, Fl_Widget::do_callback()

Enumerator

<table>
<thead>
<tr>
<th>Fl_When</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_WHEN_NEVER</td>
<td>Never call the callback.</td>
</tr>
<tr>
<td>FL_WHEN_CHANGED</td>
<td>Do the callback only when the widget value changes.</td>
</tr>
<tr>
<td>FL_WHEN_NOT_CHANGED</td>
<td>Do the callback whenever the user interacts with the widget.</td>
</tr>
<tr>
<td>FL_WHEN_RELEASE</td>
<td>Do the callback when the button or key is released and the value changes.</td>
</tr>
<tr>
<td>FL_WHEN_RELEASE_ALWAYS</td>
<td>Do the callback when the button or key is released, even if the value doesn't change.</td>
</tr>
<tr>
<td>FL_WHEN_ENTER_KEY</td>
<td>Do the callback when the user presses the ENTER key and the value changes.</td>
</tr>
<tr>
<td>FL_WHEN_ENTER_KEY_ALWAYS</td>
<td>Do the callback when the user presses the ENTER key, even if the value doesn't change.</td>
</tr>
<tr>
<td>FL_WHEN_ENTER_KEY_CHANGED</td>
<td>Do callbacks whether the value changed or not, and when the ENTER key is pressed.</td>
</tr>
<tr>
<td>FL_WHEN_CLOSED</td>
<td>Do the callback when a child of Fl_Tabs is closed.</td>
</tr>
</tbody>
</table>
13.1.5 Function Documentation

13.1.5.1 fl_box()

Fl_Boxtype fl_box ( Fl_Boxtype b ) [inline]

Get the filled version of a frame.
If no filled version of a given frame exists, the behavior of this function is undefined and some random box or frame
is returned.

13.1.5.2 fl_color_cube()

Fl_Color fl_color_cube ( int r, int g, int b ) [inline]

Returns a color out of the color cube.
r must be in the range 0 to FL_NUM_RED (5) minus 1, g must be in the range 0 to FL_NUM_GREEN (8) minus 1,
b must be in the range 0 to FL_NUM_BLUE (5) minus 1.
To get the closest color to a 8-bit set of R,G,B values use:

fl_color_cube(R * (FL_NUM_RED - 1) / 255,
            G * (FL_NUM_GREEN - 1) / 255,
            B * (FL_NUM_BLUE - 1) / 255);

13.1.5.3 fl_define_FL_EMBOSSED_LABEL()

Fl_Labeltype fl_define_FL_EMBOSSED_LABEL ( ) [extern]

Initializes the internal table entry for FL_EMBOSSED_LABEL and returns its internal value.
Internal use only.

13.1.5.4 fl_define_FL_ENGRAVED_LABEL()

Fl_Labeltype fl_define_FL_ENGRAVED_LABEL ( ) [extern]

Initializes the internal table entry for FL_ENGRAVED_LABEL and returns its internal value.
Internal use only.

13.1.5.5 fl_define_FL_ICON_LABEL()

Fl_Labeltype fl_define_FL_ICON_LABEL ( ) [extern]

Initializes the internal table entry for FL_ICON_LABEL and returns its internal value.
Internal use only.

13.1.5.6 fl_define_FL_IMAGE_LABEL()

Fl_Labeltype fl_define_FL_IMAGE_LABEL ( ) [extern]

Initializes the internal table entry for FL_IMAGE_LABEL and returns its internal value.
Internal use only.

13.1.5.7 fl_define_FL_MULTI_LABEL()

Fl_Labeltype fl_define_FL_MULTI_LABEL ( ) [extern]

Initializes the internal table entry for FL_MULTI_LABEL and returns its internal value.
Internal use only.

13.1.5.8 fl_define_FL_SHADOW_LABEL()

Fl_Labeltype fl_define_FL_SHADOW_LABEL ( ) [extern]

Initializes the internal table entry for FL_SHADOW_LABEL and returns its internal value.
Internal use only.
13.1.5.9  fl_down()

**Fl_Boxtype fl_down (Fl_Boxtype b)** [inline]

Get the "pressed" or "down" version of a box. If no "down" version of a given box exists, the behavior of this function is undefined and some random box or frame is returned.

13.1.5.10  fl_frame()

**Fl_Boxtype fl_frame (Fl_Boxtype b)** [inline]

Get the unfilled, frame only version of a box. If no frame version of a given box exists, the behavior of this function is undefined and some random box or frame is returned.

13.1.5.11  fl_gray_ramp()

**Fl_Color fl_gray_ramp (int i)** [inline]

Returns a gray color value from black (i == 0) to white (i == FL_NUM_GRAY - 1). FL_NUM_GRAY is defined to be 24 in the current FLTK release. To get the closest FLTK gray value to an 8-bit grayscale color I use:

```
fl_gray_ramp(I * (FL_NUM_GRAY - 1) / 255)
```

13.1.6 Variable Documentation

13.1.6.1  FL_ALIGN_LEFT

**const Fl_Align FL_ALIGN_LEFT = 0x0004**

Align the label at the left of the widget. Inside labels appear left-justified starting at the left side of the widget, outside labels are right-justified and drawn to the left of the widget.

13.1.6.2  FL_ALIGN_TOP

**const Fl_Align FL_ALIGN_TOP = 0x0001**

Align the label at the top of the widget. Inside labels appear below the top, outside labels are drawn on top of the widget.

13.1.6.3  FL_NORMAL_SIZE

**Fl_Fontsize FL_NORMAL_SIZE [extern]**

normal font size

13.2 Enumerations.H

Go to the documentation of this file.

00001 //
00002 // Enumerations for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2024 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file. If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 // https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 // https://www.fltk.org/bugs.php
00015 //
00016 //
#ifndef Fl_Enumerations_H
#define Fl_Enumerations_H

/***********************************************************/
* FL_ABI_VERSION is defined by configure or CMake since FLTK 1.3.4.
* It is written to FL/fl_config.h and included here.
***********************************************************/
* For more informations on FL_ABI_VERSION see README.abi-version.txt.
/***********************************************************/

#include <FL/fl_config.h>

#include "Fl_Export.H"
#include "fl_types.h"
#include <FL/platform_types.h> // for FL_COMMAND and FL_CONTROL

#define FL_MAJOR_VERSION 1
#define FL_MINOR_VERSION 4
#define FL_PATCH_VERSION 0

#define FL_VERSION ( (double)FL_MAJOR_VERSION +
                  (double)FL_MINOR_VERSION * 0.01 +
                  (double)FL_PATCH_VERSION * 0.0001 )

#define FL_API_VERSION (FL_MAJOR_VERSION*10000 + FL_MINOR_VERSION*100 + FL_PATCH_VERSION)

#ifndef FL_ABI_VERSION
#define FL_ABI_VERSION (FL_MAJOR_VERSION*10000 + FL_MINOR_VERSION*100)
#endif

/* Check if FL_ABI_VERSION is out of allowed range; redefine if necessary.
   This is done to prevent users from defining an illegal ABI version.
   Rule: FL_MAJOR_VERSION * 10000 + FL_MINOR_VERSION * 100
         <= FL_ABI_VERSION <= FL_API_VERSION.
   Example (FLTK 1.3.4):
   10300 <= FL_ABI_VERSION <= 10304
   Note: configure + CMake can be used to define FL_ABI_VERSION, but they
do not check validity. This is done here.

#define FL_ABI_VERSION < FL_MAJOR_VERSION*10000 + FL_MINOR_VERSION*100

enum Fl_Event { // events
    FL_NO_EVENT = 0,
    FL_PUSH = 1,
    FL_RELEASE = 2,
    FL_ENTER = 3,
    FL_LEAVE = 4,
    FL_DRAG = 5,
    FL_FOCUS = 6,
    FL_UNFOCUS = 7,
    FL_KEYDOWN = 8,
    FL_KEYBOARD = 8,
    FL_KEYUP = 9,
    FL_KEYDOWN = 8,
    FL_KEYBOARD = 8,
    FL_KEYUP = 9,
}
00302 FL_CLOSE = 10,
00303 00311 FL_MOVE = 11,
00324 FL_SHORTCUT = 12,
00325 00331 FL_DEACTIVATE = 13,
00332 FL_ACTIVATE = 14,
00337 00336 FL_SHORTCUT = 12,
00344 FL_DEACTIVATE = 13,
00345 00345 FL_ACTIVATE = 14,
00353 00358 FL_HIDE = 15,
00358 FL_SHOW = 16,
00359 00345 FL_PASTE = 17,
00365 FL_SELECTIONCLEAR = 18,
00370 FL_MOUSEWHEEL = 19,
00371 FL_DND_ENTER = 20,
00372 FL_DND_DRAG = 21,
00378 FL_DND_LEAVE = 22,
00384 FL_DND_RELEASE = 23,
00392 FL_SCREEN_CONFIGURATION_CHANGED = 24,
00400 FL_FULLSCREEN = 25,
00405 FL_ZOOM_GESTURE = 26,
00410 FL_ZOOM_EVENT = 27
00411 // DEV NOTE: Keep this list in sync with FL/names.h
00412
00426 enum Fl_When { // Fl_Widget::when():
00427 FL_WHEN_NEVER = 0,
00430 FL_WHEN_CHANGED = 1,
00432 FL_WHEN_NOT_CHANGED = 2,
00433 FL_WHEN_RELEASE = 4,
00434 FL_WHEN_RELEASE_ALWAYS = 6,
00435 FL_WHEN_ENTER_KEY = 8,
00437 FL_WHEN_ENTER_KEY_ALWAYS = 10,
00438 FL_WHEN_ENTER_KEY_CHANGED = 11,
00440 FL_WHEN_CLOSED = 16
00442
00453 // group: When Conditions
00458
00466 enum Fl_Callback_Reason { //FIXME: These codes collide with valid Unicode keys
00467 FL_REASON_UNKNOWN = 0,
00473 FL_REASON_SELECTED,
00474 FL_REASON_DESELECTED,
00478 FL_REASON_RESELECTED,
00480 FL_REASON_OPENED,
00481 FL_REASON_CLOSED,
00486 FL_REASON_DRAGGED,
00487 FL_REASON_CANCELLLED,
00490 FL_REASON_CHANGED,
00491 FL_REASON_GOT_FOCUS,
00494 FL_REASON_LOST_FOCUS,
00495 FL_REASON_RELEASE,
00498 FL_REASON_ENTER_KEY,
00500 FL_REASON_USER = 32
00501
00509 // group: Callback Reasons
00514
00525 //FIXME: These codes collide with valid Unicode keys
00526
00541 #define FL_Button 0xfe8
00543 #define FL_BackSpace 0xff08
00545 #define FL_Tab 0xff09
00547 #define FL_Iso_Key 0xff0c
00551 #define FL_Enter 0xff10
00556 #define FL_Pause 0xff13
00559 #define FL_Cancel 0xff14
00562 #define FL_Escape 0xff1b
00565 #define FL_Kana 0xff2e
00570 #define FL_Risu 0xff2f
00575 #define FL_Yen 0xff30
00578 #define FL_JIS_Underscore 0xff31
00582 #define FL_Home 0xff50
00586 #define FL_Left 0xff51
00590 #define FL_Up 0xff52
00594 #define FL_Right 0xff53
00595 #define FL_Down 0xff54
00600 #define FL_Page_Up 0xff55
00604 #define FL_Page_Down 0xff56
00608 #define FL_End 0xff57

Generated by Doxygen
#define FL_Print 0xff61
#define FL_Insert 0xff63
#define FL_Menu 0xff67
#define FL_Help 0xff68
#define FL_Num_Lock 0xff7f
#define FL_KP 0xff80
#define FL_KP_Enter 0xff8d
#define FL_KP_Last 0xffbd
#define FL_F 0xffbd
#define FL_F_Last 0xffe0
#define FL_Shift_L 0xffe1
#define FL_Shift_R 0xffe2
#define FL_Control_L 0xffe3
#define FL_Control_R 0xffe4
#define FL_Caps_Lock 0xffe5
#define FL_Meta_L 0xffe7
#define FL_Meta_R 0xffe8
#define FL_Alt_L 0xffe9
#define FL_Alt_R 0xffea
#define FL_Delete 0xffff

// These use the Private Use Area (PUA) of the Basic Multilingual Plane
// of Unicode. Guaranteed not to conflict with a proper Unicode character.

// These primarily map to the XFree86 keysym range
#define FL_Volume_Down 0xEF11
#define FL_Volume_Mute 0xEF12
#define FL_Volume_Up 0xEF13
#define FL_Media_Play 0xEF14
#define FL_Media_Stop 0xEF15
#define FL_Media_Prev 0xEF16
#define FL_Media_Next 0xEF17
#define FL_Home_Page 0xEF18
#define FL_Mail 0xEF19
#define FL_Search 0xEF1A
#define FL_Back 0xEF26
#define FL_Forward 0xEF27
#define FL_Stop 0xEF28
#define FL_Refresh 0xEF29
#define FL_Sleep 0xEF2A
#define FL_Favorites 0xEF2B
#define FL_LEFT_MOUSE 1
#define FL_MIDDLE_MOUSE 2
#define FL_RIGHT_MOUSE 3
#define FL_SHIFT 0x00010000
#define FL_CAPS_LOCK 0x00020000
#define FL_CTRL 0x00040000
#define FL_ALT 0x00080000
#define FL_NUM_LOCK 0x00100000
#define FL_META 0x00400000
#define FL_SCROLL_LOCK 0x00800000
#define FL_BUTTON1 0x01000000
#define FL_BUTTON2 0x02000000
#define FL_BUTTON3 0x04000000
#define FL_BUTTONS 0x7f000000
#define FL_KEY_MASK 0x0000ffff

enum Fl_Boxtype { // boxtypes (if you change these you must also change fl_boxtype.cxx):
  FL_NO_BOX = 0,
  FL_FLAT_BOX,
  FL_UP_BOX,
  FL_DOWN_BOX,
  FL_UP_FRAME,
  FL_DOWN_FRAME,
  FL_THIN_UP_BOX,
  FL_THIN_DOWN_BOX,
  FL_THIN_UP_FRAME,
  FL_THIN_DOWN_FRAME,
  FL_ENGRAVED_BOX,
FL_EMBOSSED_BOX,
FL_ENGRAVED_FRAME,
FL_EMBOSSED_FRAME,
FL_BORDER_BOX,
FL_SHADOW_BOX,
FL_BORDER_FRAME,
FL_SHADOW_FRAME,
FL_ROUND_UP_BOX,
FL_ROUND_DOWN_BOX,
FL_DIAMOND_UP_BOX,
FL_DIAMOND_DOWN_BOX,
FL_OVAL_BOX,
FL_OSHADOW_BOX,
FL_OVAL_FRAME,
FL_OFLAT_BOX,
FL_PLASTIC_UP_BOX,
FL_PLASTIC_DOWN_BOX,
FL_PLASTIC_UP_FRAME,
FL_PLASTIC_DOWN_FRAME,
#define FL_PLASTIC_UP_FRAME (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+2)
#define FL_PLASTIC_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+3)
#define FL_PLASTIC_THIN_UP_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+4)
#define FL_PLASTIC_THIN_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+5)
#define FL_PLASTIC_ROUND_UP_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+6)
#define FL_PLASTIC_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+7)

extern FL_EXPORT Fl_Boxtype fl_define_FL_GTK_UP_BOX();
#define FL_GTK_UP_BOX fl_define_FL_GTK_UP_BOX()
#define FL_GTK_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+1)
#define FL_GTK_UP_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+2)
#define FL_GTK_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+3)
#define FL_GTK_THIN_UP_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+4)
#define FL_GTK_THIN_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+5)
#define FL_GTK_THIN_UP_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+6)
#define FL_GTK_THIN_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+7)
#define FL_GTK_ROUND_UP_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+8)
#define FL_GTK_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+9)

extern FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX();
#define FL_GLEAM_UP_BOX fl_define_FL_GLEAM_UP_BOX()
#define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+1)
#define FL_GLEAM_UP_FRAME (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+2)
#define FL_GLEAM_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+3)
#define FL_GLEAM_THIN_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+4)
#define FL_GLEAM_THIN_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+5)
#define FL_GLEAM_ROUND_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+6)
#define FL_GLEAM_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+7)

extern FL_EXPORT Fl_Boxtype fl_define_FL_OXY_UP_BOX();
#define FL_OXY_UP_BOX fl_define_FL_OXY_UP_BOX()
#define FL_OXY_DOWN_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+1)
#define FL_OXY_UP_FRAME (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+2)
#define FL_OXY_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+3)
#define FL_OXY_THIN_UP_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+4)
#define FL_OXY_THIN_DOWN_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+5)
#define FL_OXY_THIN_UP_FRAME (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+6)
#define FL_OXY_THIN_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+7)
#define FL_OXY_ROUND_UP_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+8)
#define FL_OXY_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+9)
#define FL_OXY_BUTTON_UP_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+10)
#define FL_OXY_BUTTON_DOWN_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+11)

#endif // ! FL_DOXYGEN

// conversions of box types to other box types:
inline Fl_Boxtype fl_box(Fl_Boxtype b) {
    return (Fl_Boxtype)((b<FL_UP_BOX||b%4>1)?b:(b-2));
}
inline Fl_Boxtype fl_down(Fl_Boxtype b) {
    return (Fl_Boxtype)((b<FL_UP_BOX)?b:(b|1));
}
inline Fl_Boxtype fl_frame(Fl_Boxtype b) {
    return (Fl_Boxtype)((b%4<2)?b:(b+2));
}

// back-compatibility box types:
#define FL_FRAME FL_ENGRAVED_FRAME
#define FL_FRAME_BOX FL_ENGRAVED_BOX
#define FL_CIRCLE_BOX FL_ROUND_DOWN_BOX
#define FL_DIAMOND_BOX FL_DIAMOND_DOWN_BOX

// group: Box Types
enum Fl_Labeltype { // labeltypes:
    FL_NORMAL_LABEL = 0,
    FL_NO_LABEL,
    FL_SHADOW_LABEL,
    FL_ENGRAVED_LABEL,
    FL_EMBossed_LABEL,
    FL_MULTI_LABEL,
    FL_ICON_LABEL,
    FL_IMAGE_LABEL,
    FL_FREE_LABELTYPE
};
13.2 Enumerations.H 1995

```c
#define FL_MULTI_LABEL fl_define_FL_MULTI_LABEL()

extern Fl_Labeltype FL_EXPORT fl_define_FL_ICON_LABEL();
#define FL_ICON_LABEL fl_define_FL_ICON_LABEL()

extern Fl_Labeltype FL_EXPORT fl_define_FL_IMAGE_LABEL();
#define FL_IMAGE_LABEL fl_define_FL_IMAGE_LABEL()

typedef unsigned Fl_Align;
const Fl_Align FL_ALIGN_CENTER = 0x0000;
const Fl_Align FL_ALIGN_TOP = 0x0001;
const Fl_Align FL_ALIGN_BOTTOM = 0x0002;
const Fl_Align FL_ALIGN_LEFT = 0x0004;
const Fl_Align FL_ALIGN_RIGHT = 0x0008;
const Fl_Align FL_ALIGN_INSIDE = 0x0010;
const Fl_Align FL_ALIGN_TEXT_OVER_IMAGE = 0x0020;
const Fl_Align FL_ALIGN_IMAGE_OVER_TEXT = 0x0000;
const Fl_Align FL_ALIGN_CLIP = 0x0040;
const Fl_Align FL_ALIGN_WRAP = 0x0080;
const Fl_Align FL_ALIGN_IMAGE_NEXT_TO_TEXT = 0x0100;
const Fl_Align FL_ALIGN_TEXT_NEXT_TO_IMAGE = 0x0120;
const Fl_Align FL_ALIGN_IMAGE_BACKDROP = 0x0200;
const Fl_Align FL_ALIGN_LEFT_TOP = FL_ALIGN_TOP | FL_ALIGN_LEFT;
const Fl_Align FL_ALIGN_TOP_RIGHT = FL_ALIGN_TOP | FL_ALIGN_RIGHT;
const Fl_Align FL_ALIGN_BOTTOM_LEFT = FL_ALIGN_BOTTOM | FL_ALIGN_LEFT;
const Fl_Align FL_ALIGN_BOTTOM_RIGHT = FL_ALIGN_BOTTOM | FL_ALIGN_RIGHT;
const Fl_Align FL_ALIGN_NOWRAP = 0x0000;
const Fl_Align FL_ALIGN_POSITION_MASK = 0x000f;
const Fl_Align FL_ALIGN_IMAGE_MASK = 0x320;

typedef int Fl_Font;
const Fl_Font FL_HELVETICA = 0;
const Fl_Font FL_HELVETICA_BOLD = 1;
const Fl_Font FL_HELVETICA_ITALIC = 2;
const Fl_Font FL_HELVETICA_BOLD_ITALIC = 3;
const Fl_Font FL_COURIER = 4;
const Fl_Font FL_COURIER_BOLD = 5;
const Fl_Font FL_COURIER_ITALIC = 6;
const Fl_Font FL_COURIER_BOLD_ITALIC = 7;
const Fl_Font FL_TIMES = 8;
const Fl_Font FL_TIMES_BOLD = 9;
const Fl_Font FL_TIMES_ITALIC = 10;
const Fl_Font FL_TIMES_BOLD_ITALIC = 11;
const Fl_Font FL_SYMBOL = 12;
const Fl_Font FL_SCREEN = 13;
const Fl_Font FL_ZAPF_DINGBATS = 14;
const Fl_Font FL_FREE_FONT = 16;
const Fl_Font FL_BOLD = 1;
const Fl_Font FL_ITALIC = 2;
const Fl_Font FL_BOLD_ITALIC = 3;

typedef int Fl_Fontsize;
extern FL_EXPORT Fl_Fontsize FL_NORMAL_SIZE;

typedef unsigned int Fl_Color;
const Fl_Color FL_FOREGROUND_COLOR = 0;
const Fl_Color FL_BACKGROUND2_COLOR = 7;
```

// Standard colors. These are used as default colors in widgets and altered as necessary
const Fl_Color FL_FOREGROUND_COLOR = 0;
const Fl_Color FL_BACKGROUND2_COLOR = 7;

Generated by Doxygen
const Fl_Color FL_INACTIVE_COLOR = 8;
const Fl_Color FL_SELECTION_COLOR = 15;

// boxtypes generally limit themselves to these colors so
// the whole ramp is not allocated:
const Fl_Color FL_GRAY0 = 32; // 'A'
const Fl_Color FL_DARK3 = 39; // 'H'
const Fl_Color FL_DARK2 = 45; // 'N'
const Fl_Color FL_DARK1 = 47; // 'P'
const Fl_Color FL_BACKGROUND_COLOR = 49; // 'R' default background color
const Fl_Color FL_LIGHT1 = 50; // 'S'
const Fl_Color FL_LIGHT2 = 52; // 'U'
const Fl_Color FL_LIGHT3 = 54; // 'W'

// FLTK provides a 5x8x5 color cube that is used with colormap visuals
const Fl_Color FL_BLACK = 56;
const Fl_Color FL_RED = 88;
const Fl_Color FL_GREEN = 63;
const Fl_Color FL_YELLOW = 95;
const Fl_Color FL_BLUE = 216;
const Fl_Color FL_MAGENTA = 248;
const Fl_Color FL_CYAN = 223;
const Fl_Color FL_DARK_RED = 72;
const Fl_Color FL_DARK_GREEN = 60;
const Fl_Color FL_DARK_YELLOW = 76;
const Fl_Color FL_DARK_BLUE = 136;
const Fl_Color FL_DARK_MAGENTA = 152;
const Fl_Color FL_DARK_CYAN = 140;
const Fl_Color FL_WHITE = 255;

#define FL_FREE_COLOR (Fl_Color)16
#define FL_NUM_FREE_COLOR 16
#define FL_GRAY_RAMP (Fl_Color)32
#define FL_NUM_GRAY 24
#define FL_GRAY FL_BACKGROUND_COLOR
#define FL_COLOR_CUBE (Fl_Color)56
#define FL_NUM_RED 5
#define FL_NUM_GREEN 8
#define FL_NUM_BLUE 5

FL_EXPORT Fl_Color fl_inactive(Fl_Color c);

typedef Fl_Color (Fl_Contrast_Function)(Fl_Color, Fl_Color, Fl_Fontsize, int);

FL_EXPORT void fl_contrast_function(Fl_Contrast_Function *f);

enum Fl_Contrast_Mode {
    FL_CONTRAST_NONE = 0,
    FL_CONTRAST_LEGACY,
    FL_CONTRAST_CIELAB,
    FL_CONTRAST_CUSTOM,
    FL_CONTRAST_LAST
};

FL_EXPORT void fl_contrast_level(int level);
FL_EXPORT int fl_contrast_level();
FL_EXPORT void fl_contrast_mode(int mode);
FL_EXPORT int fl_contrast_mode();

FL_EXPORT Fl_Color fl_contrast(Fl_Color fg, Fl_Color bg, Fl_Fontsize fs = 0, int context = 0);
FLEXPORT double fl_lightness(Fl_Color color);
FL_EXPORT double fl_luminance(Fl_Color color);

enum Fl_Contrast_Mode {
    FL_CONTRAST_NONE = 0,
    FL_CONTRAST_LEGACY,
    FL_CONTRAST_CIELAB,
    FL_CONTRAST_CUSTOM,
    FL_CONTRAST_LAST
};

// Other color functions are defined and documented in src/fl_color.cxx

FL_EXPORT void fl_contrast_level(int level);
FL_EXPORT int fl_contrast_level();
FL_EXPORT void fl_contrast_mode(int mode);
FL_EXPORT int fl_contrast_mode();

FL_EXP Fl_Color fl_contrast(Fl_Color fg, Fl_Color bg, Fl_Fontsize fs = 0, int context = 0);
01220
01221 \texttt{inline Fl\_Color fl\_gray\_ramp(int i) \{ return (Fl\_Color)(i+FL\_GRAY\_RAMP);\}}
01222
01223 \texttt{inline Fl\_Color fl\_color\_cube(int r, int g, int b) \{
01224 \quad \text{return (Fl\_Color)((b*FL\_NUM\_RED + r) * FL\_NUM\_GREEN + g + FL\_COLOR\_CUBE));\}}
01225
01226 // group: Colors 01251
01227 * FIXME: We should renumber these, but that will break the ABI */
01228 \texttt{enum Fl\_Cursor \{ 01229 \quad FL\_CURSOR\_DEFAULT = 0, // U+2196 01230 \quad FL\_CURSOR\_ARROW = 35, // U+2196 01231 \quad FL\_CURSOR\_CROSS = 66, // U+FF0B 01232 \quad FL\_CURSOR\_WAIT = 76, // U+231A, U+231B 01233 \quad FL\_CURSOR\_INSERT = 77, // U+2316 01234 \quad FL\_CURSOR\_HAND = 31, // U+261C 01235 \quad FL\_CURSOR\_HELP = 47,
01236 \quad FL\_CURSOR\_MOVE = 27, // U+2725, U+270B 01237 \};
01238 */ * Resize indicators */
01239 \texttt{FL\_CURSOR\_NS = 78, // U+21D5 01240 \quad FL\_CURSOR\_NW = 79, // U+21D4 01241 \quad FL\_CURSOR\_MSW = 80, // U+2921 01242 \quad FL\_CURSOR\_NESW = 81, // U+2922 01243 \quad FL\_CURSOR\_N \quad = 64, // U+2912 01244 \quad FL\_CURSOR\_NE \quad = 65, // U+2197 01245 \quad FL\_CURSOR\_R \quad = 49, // U+21E5 01246 \quad FL\_CURSOR\_SE \quad = 8, // U+21F2 01247 \quad FL\_CURSOR\_S \quad = 31, // U+261C 01248 \quad FL\_CURSOR\_SW \quad = 7, // U+2199 01249 \quad FL\_CURSOR\_W \quad = 36, // U+21E4 01250 \quad FL\_CURSOR\_NW = 68, // U+21F1 01251 \quad FL\_CURSOR\_NONE \quad =255 01252 \};
01253 // group: Cursors 01288 01254 \texttt{enum \{ // values for "when" passed to Fl::add_fd() 01255 \quad FL\_READ = 1, 01256 \quad FL\_WRITE = 4, 01257 \quad FL\_EXCEPT = 8 01258 \};
01259 \};
01260 \texttt{enum Fl\_Mode \{ 01261 \quad FL\_RGB = 0, 01262 \quad FL\_INDEX = 1, 01263 \quad FL\_SINGLE = 0, 01264 \quad FL\_DOUBLE = 2, 01265 \quad FL\_ACTUAL = 4, 01266 \quad FL\_ALPHA = 8, 01267 \quad FL\_DEPTH = 16, 01268 \quad FL\_STENCIL = 32, 01269 \quad FL\_RGB8 = 64, 01270 \quad FL\_MULTISAMPLE = 128, 01271 \quad FL\_STEREO = 256, 01272 \quad FL\_FAKE\_SINGLE = 512, // Fake single buffered windows using double-buffer 01273 \quad FL\_OPENCGL = 1024 01274 \};
01275 // image alpha blending 01276 \texttt{#define FL\_IMAGE\_WITH\_ALPHA 0x40000000 01277 \};
01278 \texttt{enum Fl\_Damage \{ 01279 \quad FL\_ DAMAGE\_CHILD = 0x01, 01280 \quad FL\_ DAMAGE\_EXPOSE = 0x02, 01281 \quad FL\_ DAMAGE\_SCROLL = 0x04, 01282 \quad FL\_ DAMAGE\_OVERLAY = 0x08, 01283 \quad FL\_ DAMAGE\_USER1 = 0x10, 01284 \quad FL\_ DAMAGE\_USER2 = 0x20, 01285 \quad FL\_ DAMAGE\_ALL = 0x80 01286 \};
01287 */ 01288 \texttt{enum Fl\_Arrow\_Type \{ 01289 \quad FL\_ARROW\_SINGLE = 0x01, 01290 \quad FL\_ARROW\_DOUBLE = 0x02, 01291 \quad FL\_ARROW\_SINGLE = 0x03, 01292 \quad FL\_ARROW\_RETURN = 0x04 01293 \};
01294 \};
01295 \texttt{enum Fl\_Orientation \{ 01296 \quad FL\_ORIENT\_NONE = 0x00, 01297 \quad FL\_ORIENT\_RIGHT = 0x05, 01298 \quad FL\_ORIENT\_HE = 0x01, 01299 \quad FL\_ORIENT\_UP = 0x02, 01300 \quad FL\_ORIENT\_NW = 0x03, 01301 \quad FL\_ORIENT\_LEFT = 0x04, 01302 \}}
13.3  filename.H File Reference

File names and URI utility functions.
#include "Fl_Export.H"
#include <FL/platform_types.h>

Macros

• #define FL_PATH_MAX 2048
   all path buffers should use this length

Typedefs

• typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)
  File sorting function.

Functions

• void fl_decode_uri (char *uri)
  Decodes a URL-encoded string.
• int fl_filename_absolute (char *to, int tolen, const char *from)
  Makes a filename absolute from a relative filename to the current working directory.
• int fl_filename_absolute (char *to, int tolen, const char *from, const char *cwd)
  Concatenate the absolute path base with from to form the new absolute path in to.
• int fl_filename_expand (char *to, int tolen, const char *from)
  Expands a filename containing shell variables and tilde (~).
• const char * fl_filename_ext (const char *buf)
  Gets the extension of a filename.
• void fl_filename_free_list (struct dirent ***, int n)
  Free the list of filenames that is generated by fl_filename_list().
• int fl_filename_isdir (const char *name)
  Determines if a file exists and is a directory from its filename.
• int fl_filename_list (const char *d, struct dirent ***, Fl_File_Sort_F *s=fl_numericsort)
  Portable and const-correct wrapper for the scandir() function.
• int fl_filename_match (const char *name, const char *pattern)
  Checks if a string s matches a pattern p.
• const char * fl_filename_name (const char *filename)
  Gets the file name from a path.
• int fl_filename_relative (char *to, int tolen, const char *from)
  Makes a filename relative to the current working directory.
• int fl_filename_relative (char *to, int tolen, const char *from, const char *cwd)
  Makes a filename relative to any other directory.
• char * fl_filename_setext (char *to, int tolen, const char *ext)
  Replaces the extension in buf of max.
• int fl_open_uri (const char *uri, char *msg, int msglen)
  Opens the specified Uniform Resource Identifier (URI).
13.3.1 Detailed Description

File names and URI utility functions.

13.4 filename.H

Go to the documentation of this file.

```
/*
 * Filename header file for the Fast Light Tool Kit (FLTK).
 *
 * Copyright 1998-2023 by Bill Spitzak and others.
 *
 * This library is free software. Distribution and use rights are outlined in
 * the file "COPYING" which should have been included with this file. If this
 * file is missing or damaged, see the license at:
 * https://www.fltk.org/COPYING.php
 *
 * Please see the following page on how to report bugs and issues:
 * https://www.fltk.org/bugs.php
 */

#ifndef FL_FILENAME_H
#define FL_FILENAME_H

#include "Fl_Export.H"
#include <FL/platform_types.h>

#ifdef __cplusplus
  // The following include is not (yet) used in FLTK 1.4
  // In FLTK 1.5 or 4.0 using std::string would be default.
  #include <string>
#endif /* __cplusplus */

#define FL_PATH_MAX 2048

FL_EXPORT const char *fl_filename_name(const char * filename);
FL_EXPORT const char *fl_filename_ext(const char *buf);
FL_EXPORT char *fl_filename_setext(char *to, int tolen, const char *ext);
FL_EXPORT int fl_filename_expand(char *to, int tolen, const char *from);
FL_EXPORT int fl_filename_absolute(char *to, int tolen, const char *from);
FL_EXPORT int fl_filename_relative(char *to, int tolen, const char *from);
FL_EXPORT int fl_filename_match(const char *name, const char *pattern);
FL_EXPORT int fl_filename_isdir(const char *name);

#ifdef __cplusplus
  FL_EXPORT int fl_filename_absolute(char *to, int tolen, const char *from, const char *cwd);
  FL_EXPORT int fl_filename_relative(char *to, int tolen, const char *from, const char *base);
  // FIXME: We can't do this in 1.4.x - enable this block in 1.5 or higher.
  // See fluid/fluid_filename.{h|cxx} for an implementation using Fl_String.
  FL_EXPORT std::string fl_filename_name(const std::string &filename);
  FL_EXPORT std::string fl_filename_path(const std::string &filename);
  FL_EXPORT std::string fl_filename_ext(const std::string &filename);
  FL_EXPORT std::string fl_filename_setext(const std::string &filename, const std::string &new_extension);
  FL_EXPORT std::string fl_filename_expand(const std::string &from);
  FL_EXPORT std::string fl_filename_relative(const std::string &from, const std::string &base);
  FL_EXPORT std::string fl_filename_absolute(const std::string &from, const std::string &base);
  // FL_EXPORT std::string fl_getcwd();
#endif /* defined(__cplusplus) */

#ifndef __cplusplus
  #if defined(__cplusplus)
    FL_EXPORT int fl_filename_absolute(char *to, const char *name, const char * cwd);
    FL_EXPORT int fl_filename_relative(char *to, const char *name);
    #endif
  #endif
#endif

#define __cplusplus

inline char *fl_filename_setext(char *to, const char *ext) { return fl_filename_setext(to, FL_PATH_MAX, ext); }
```
#include <FL/fl_config.h>
#include <FL/Fl_Export.H>
#include <FL/platform_types.h>
#include <FL/fl_casts.H>
#include <FL/Fl_Cairo.H>
#include "fl_utf8.h"
#include "Enumerations.H"
#include <string.h>

## 13.5 Fl.H File Reference

**Fl** static class.

```c
#include <FL/fl_config.h>
#include <FL/Fl_Export.H>
#include <FL/platform_types.h>
#include <FL/fl_casts.H>
#include <FL/Fl_Cairo.H>
#include "fl_utf8.h"
#include "Enumerations.H"
#include <string.h>
```

### Classes

- **class Fl**
  
  The **Fl** is the FLTK global (static) class containing state information and global methods for the current application.

- **class Fl_Widget_Tracker**
  
  This class should be used to control safe widget deletion.

### Macros

- `#define Fl_Object Fl_Widget`
for back compatibility - use Fl_Widget!

**Typedefs**

- typedef void(\* Fl_Abort_Handler) (const char \*format,...)  
  Signature of set_abort functions passed as parameters.
- typedef int(\* Fl_Args_Handler) (int argc, char \*\*argv, int &i)  
  Signature of args functions passed as parameters.
- typedef void(\* Fl_Atclose_Handler) (Fl_Window \*window, void \*data)  
  Signature of set_atclose functions passed as parameters.
- typedef void(\* Fl_Awake_Handler) (void \*data)  
  Signature of some wakeup callback functions passed as parameters.
- typedef void() Fl_Box_Draw_F(int x, int y, int w, int h, Fl_Color color)  
  Signature of some box drawing functions passed as parameters.
- typedef void(\* Fl_Clipboard_Notify_Handler) (int source, void \*data)  
  Signature of add_clipboard_notify functions passed as parameters.
- typedef int(\* Fl_Event_Dispatch) (int event, Fl_Window \*w)  
  Signature of event_dispatch functions passed as parameters.
- typedef int(\* Fl_Event_Handler) (int event)  
  Signature of add_handler functions passed as parameters.
- typedef void(\* Fl_FD_Handler) (FL_SOCKET fd, void \*data)  
  Signature of add_fd functions passed as parameters.
- typedef void(\* Fl_Idle_Handler) (void \*data)  
  Signature of add_idle callback functions passed as parameters.
- typedef void() Fl_Label_Draw_F(const Fl_Label \*label, int x, int y, int w, int h, Fl_Align align)  
  Signature of some label drawing functions passed as parameters.
- typedef void() Fl_Label_Measure_F(const Fl_Label \*label, int \&width, int \&height)  
  Signature of some label measurement functions passed as parameters.
- typedef void(\* Fl_Old_Idle_Handler) ()  
  Signature of set_idle callback functions passed as parameters.
- typedef int(\* Fl_System_Handler) (void \*event, void \*data)  
  Signature of add_system_handler functions passed as parameters.
- typedef void(\* Fl_Timeout_Handler) (void \*data)  
  Signature of timeout callback functions passed as parameters.

**Variables**

- const char \* fl_local_alt  
  string pointer used in shortcuts, you can change it to another language
- const char \* fl_local_ctrl  
  string pointer used in shortcuts, you can change it to another language
- const char \* fl_local_meta  
  string pointer used in shortcuts, you can change it to another language
- const char \* fl_local_shift  
  string pointer used in shortcuts, you can change it to another language

### 13.5.1 Detailed Description

Fl static class.
FILE DOCUMENTATION

13.6 FL.H

Go to the documentation of this file.

13.6 FL.H

// Main header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2023 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file COPYING which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#ifndef Fl_H
#define Fl_H

#include <FL/fl_config.h> // build configuration
#include <FL/Fl_Export.H>
#include <FL/platform_types.h> // for FL_SOCKET
#include <FL/fl_casts.H> // experimental

#ifdef FLTK_HAVE_CAIRO
#include <FL/Fl_Cairo.H>
#endif

#include "fl_utf8.h"
#include "Enumerations.H"
ifndef Fl_Object
#define Fl_Object Fl_Widget
#endif

define check
#undef check

define BSD
#undef BSD

#include <string.h> // FIXME: Fl::is_scheme(): strcmp needs string.h

class Fl_Widget;
class Fl_Window;
class Fl_Image;
struct Fl_Label;
class Fl_Screen_Driver;
class Fl_System_Driver;

// Pointers you can use to change FLTK to another language.
// Note: Similar pointers are defined in FL/fl_ask.H and src/fl_ask.cxx

typedef void (FL_Label_Draw_F)(const Fl_Label *label, int x, int y, int w, int h, Fl_Align align);
typedef void (FL_Label_Measure_F)(const Fl_Label *label, int &width, int &height);
typedef void (FL_Box_Draw_F)(int x, int y, int w, int h, Fl_Color color);
typedef void (*Fl_Timeout_Handler)(void *data);
typedef void (*Fl_Awake_Handler)(void *data);
typedef void (*Fl_Idle_Handler)(void *data);
typedef void (*Fl_Old_Idle_Handler)();
typedef void (*Fl_FD_Handler)(FL_SOCKET fd, void *data);
typedef int (*Fl_Event_Handler)(int event);
typedef int (*Fl_System_Handler)(void *event, void *data);
typedef void (*Fl_Abort_Handler)(const char *format,...);
typedef void (*Fl_Atclose_Handler)(Fl_Window *window, void *data);
typedef int (*Fl_Args_Handler)(int argc, char **argv, int &i);

typedef int (*Fl_Event_Dispatch)(int event, Fl_Window *w);

typedef void (*Fl_Clipboard_Notify_Handler)(int source, void *data);

/* group callback_functions */

class FL_EXPORT Fl {
    friend class Fl_System_Driver;
    Fl(); { // no constructor!

    private:

    static int use_high_res_GL;
    static int draw_GL_text_with_textures;
    static int box_shadow_width;
    static int box_border_radius_max;
    static int selection_to_clipboard;

    public:

    static Fl_Screen_Driver *screen_driver();
    static Fl_System_Driver *system_driver();

    #ifdef __APPLE__ // deprecated in 1.4 - only for compatibility with 1.3
    static void reset_marked_text();
    static void insertion_point_location(int x, int y, int height);
    #endif

    static int box_shadow_width() { return box_shadow_width_; }
    static void box_shadow_width(int W) { box_shadow_width_ = W < 1 ? 1 : W; }

    static int box_border_radius_max() { return box_border_radius_max_; }
    static void box_border_radius_max(int R) { box_border_radius_max_ = R < 5 ? 5 : R; }

    public: // should be private!

    static void damage(int d) {damage_ = d;}

    public:

    typedef enum {
        OPTION_ARROW_FOCUS = 0,
        // When switched on, FLTK will use the file chooser dialog that comes
        // with your operating system whenever possible. When switched off, FLTK
        // will present its own file chooser.
        // // TODO implement me
        // OPTION_NATIVE_FILECHOOSER,
        // When Filechooser Preview is enabled, the FLTK or native file chooser
        // will show a preview of a selected file (if possible) before the user
        // decides to choose the file.
        // // TODO implement me
        // OPTION_FILECHOOSER_PREVIEW,
        OPTION_VISIBLE_FOCUS,
        OPTION_DND_TEXT,
        OPTION_SHOW_TOOLTIPS,
        OPTION_FNFC_USES_GTK,
    }
# File Documentation

```cpp
00272 #define OPTION_PRINTER_USES_GTK,
00276 #define OPTION_SHOW_SCALING,
00279 #define OPTION_FNFC_USES_ZENITY,
00280 // don't change this, leave it always as the last element
00282 #define OPTION_LAST
00283 } Fl_Option;
00284
00285 private:
00286 static unsigned char options_[(OPTION_LAST)];
00287 static unsigned char options_read_
00288 static int program_should_quit_ // non-zero means the program was asked to cleanly terminate
00289
00290 public:
00291 /*
00292 Return a global setting for all FLTK applications, possibly overridden
00293 by a setting specifically for this application.
00294 */
00295 static bool option(Fl_Option opt);
00296
00297 /*
00298 Override an option while the application is running.
00299 */
00300 static void option(Fl_Option opt, bool val);
00301
00309 static ( *idle)();
00310
00311 ifndef FL_DOXYGEN
00312 private:
00313 static Fl_Awake_Handler *awake_ring_;
00314 static void **awake_data_;
00315 static int awake_ring_size_;  // late addition
00316 static int awake_ring_head_;  // late addition
00317 static int awake_ring_tail_;  // late addition
00318 public:
00319 static const char* scheme_;  // late addition
00320 static Fl_Image* scheme_bg_;  // late addition
00321 static int e_original_keysym; // late addition
00322 static int scrollbar_size_;  // STR #2927
00323 static int menu_linespacing_;  // STR #2927
00324 #endif
00325 #define FL_DOXYGEN
00326
00327 static int add_awake_handler_(Fl_Awake_Handler, void *);
00328 static int get_awake_handler_(Fl_Awake_Handler&, void *&);
00329
00332 static int api_version();
00333 static int abi_version();
00334
00337 static int abi_version();
00338 static int api_version();
00339
00342 static int is_scheme(const char *name) {
00343 return (scheme_ && name && !strcmp(name, scheme_));
00344 }
00345
00346 static int reload_scheme(); // defined in 'src/Fl_get_system_colors.cxx'
00347
00352 static const char * const help;
00353
00356 // things called by initialization:
00357 static void display(const char *);
00358 static int visual(int);
00359 static int gl_visual(int, int *alist=0); // platform dependent
00360 static void own_colormap();
00361 static void get_system_colors();
00362 static void foreground(uchar, uchar, uchar);
00363 static void background(uchar, uchar, uchar);
00364 static void background2(uchar, uchar, uchar);
00365
00367 // schemes:
00368 static int scheme(const char *name);
00369 static const char* scheme() { return scheme_; }  // late addition
00370
00371 static int is_scheme(const char *name) {
00372 return (scheme_ && name && !strcmp(name, scheme_));
00373 }
00374
00375 static int scrollbar_size();  // defined in 'src/Fl_get_system_colors.cxx'
00376 static void scrollbar_size(int W);
00377
00378 static int arg(int argc, char **argv, int& i);
00379 static int args(int argc, char **argv, int& i, Fl_Args_Handler cb = 0);
00380 static void args(int argc, char **argv);
00381
00382 static inline int abi_check(const int val = FL_ABI_VERSION) {
00383 return val == abi_version();
00384 }
00385
00386 // argument parsers:
00387 static int arg(int argc, char **argv, int& i);
00388 static int args(int argc, char **argv, int& i, Fl_Args_Handler cb = 0);
00389 static void args(int argc, char **argv);
00390 static const char* const help;
00391
00392 // schemes:
00393 static int scheme(const char *name);
00394 static const char* scheme() { return scheme_; }
00395
00396 static int is_scheme(const char *name) {
00397 return (scheme_ && name && !strcmp(name, scheme_));
00398 }
00399
00400 static int reload_scheme(); // defined in 'src/Fl_get_system_colors.cxx'
00401
00402 static int scrollbar_size();
00403 static void scrollbar_size(int W);
00404
00405 Generated by Doxygen
```
static int menu_linespacing();
static void menu_linespacing(int H);

// execution:
static int wait();
static double wait(double time);
static int check();
static int ready();
static int run();
static int program_should_quit() {return program_should_quit_;

static void program_should_quit(int should_i) { program_should_quit_ = should_i;

static void hide_all_windows();

static Fl_Widget* readqueue();

// cross-platform timer support
static void add_timeout(double t, Fl_Timeout_Handler cb, void *data = 0);
static void repeat_timeout(double t, Fl_Timeout_Handler cb, void *data = 0);
static int has_timeout(Fl_Timeout_Handler cb, void *data = 0);
static void remove_timeout(Fl_Timeout_Handler cb, void *data = 0);

static void add_check(Fl_Timeout_Handler, void * = 0);
static int has_check(Fl_Timeout_Handler, void * = 0);
static void remove_check(Fl_Timeout_Handler, void * = 0);

static Fl_Timestamp now(double offset = 0);
static double seconds_since(Fl_Timestamp& then);
static double seconds_between(Fl_Timestamp& back, Fl_Timestamp& further_back);
static long ticks_since(Fl_Timestamp& then);
static long ticks_between(Fl_Timestamp& back, Fl_Timestamp& further_back);

// private
static void run_idle();

// event information:
static int event() {return e_number;
static int event_x() {return e_x;
static int event_y() {return e_y;
static int event_x_root() {return e_x_root;
static int event_y_root() {return e_y_root;
static int event_dx() {return e_dx;
static int event_dy() {return e_dy;
static void get_mouse(int &x, int &y);
static int event_clicks() {return e_clicks;
static void event_clicks(int i) {e_clicks = i;
static int event_is_click() {return e_is_click;
static int event_is_click(int i) {e_is_click = i;
static int event_button() {return e_keysym-FL_Button;
static int event_state() {return e_state;
static int event_state(int mask) {return e_state & mask;
static void event_clipboard() { return e_clipboard_data;
static const char * event_clipboard_type() {return e_clipboard_type;

Generated by Doxygen
static void compose_reset();
static int event_inside(int, int, int, int);
static int event_inside(const Fl_Widget*);
static int test_shortcut(Fl_Shortcut);

static void enable_im();
static void disable_im();

// event destinations:
static int handle(int, Fl_Window*);
static int handle_(int, Fl_Window*);
static Fl_Widget* belowmouse() {return belowmouse_};
static void belowmouse(Fl_Widget*);
static Fl_Widget* pushed() {return pushed_};
static void pushed(Fl_Widget*);
static Fl_Widget* focus() {return focus_};
static void focus(Fl_Widget*);
static void add_handler(Fl_Event_Handler h);
static void remove_handler(Fl_Event_Handler h);
static void add_system_handler(Fl_System_Handler h, void *data);
static void remove_system_handler(Fl_System_Handler h);
static void event_dispatch(Fl_Event_Dispatch d);
static Fl_Event_Dispatch event_dispatch();
static Fl_Callback_Reason callback_reason();

// cut/paste:
static void copy(const char *, int len, int destination = 0,
                 const char *, Fl::clipboard_plain_text);

static void selection_to_clipboard(int mode) {
  selection_to_clipboard_ = mode ? 1 : 0;
}
static int selection_to_clipboard() { return selection_to_clipboard_; }

static void paste(Fl_Widget &receiver, int source, const char *type = Fl::clipboard_plain_text);

static void add_clipboard_notify(Fl_Clipboard_Notify_Handler h, void *data = 0);
static void remove_clipboard_notify(Fl_Clipboard_Notify_Handler h);
static int clipboard_contains(const char *type);
static char const * const clipboard_plain_text;
static char const * const clipboard_image;

static int dnd(); // platform dependent
static void paste(Fl_Widget &receiver);
static Fl_Callback_Reason callback_reason();

// multi-head support:
static int screen_count(); // via screen driver
static void screen_xywh(int &X, int &Y, int &W, int &H); // via screen driver
static void screen_xywh(int &X, int &Y, int &W, int &H, int mx, int my); // via screen driver
static void screen_xywh(int &X, int &Y, int &W, int &H, int n); // via screen driver
static void screen_xywh(int &X, int &Y, int &W, int &H, int mx, int my, int n); // via screen driver
static float screen_scale(int n); // via screen driver
static void screen_scale(int n, float factor); // via screen driver
static int screen_scaling_supported();
static void keyboard_screen_scaling(int value);

// color map:
static void set_color(Fl_Color, uchar, uchar, uchar);
static void set_color(Fl_Color, uchar, uchar, uchar, uchar);
static void set_color(Fl_Color i, uchar *red, uchar *green, uchar *blue);
static void get_color(Fl_Color i, uchar *red, uchar *green, uchar *blue, uchar *alpha);
static void free_color(Fl_Color i, int overlay = 0); // platform dependent
static void set_font(Fl_Font, int* attributes = 0);
static void set_font(Fl_Font, int* size);
static void set_font(Fl_Font, char*);
static void set_font(Fl_Font, Fl_Font);
1191  static Fl_Font set_fonts(const char * = 0); // platform dependent
1192}
1199  // Hack to re-order the 'Drawing functions' group
1202  // labeltypes:
1203  static void set_labeltype(Fl_Labeltype, Fl_Label_Draw_F*, Fl_Label_Measure_F*);
1205  static void set_labeltype(Fl_Labeltype, Fl_Labeltype from); // is it defined?
1206
1207  // boxtypes:
1208  static Fl_Box_Draw_F *get_boxtype(Fl_Boxtype);
1209  static void set_boxtype(Fl_Boxtype, Fl_Box_Draw_F*, uchar,uchar,uchar,uchar);
1210  static void set_boxtype(Fl_Boxtype, Fl_Boxtype from);
1211  static int box_dx(Fl_Boxtype);
1212  static int box_dy(Fl_Boxtype);
1213  static int box_dw(Fl_Boxtype);
1214  static int box_dh(Fl_Boxtype);
1215  static int draw_box_active();
1216  static Fl_Color box_color(Fl_Color);
1217  static void set_box_color(Fl_Color);
1218
1219  // Back compatibility:
1220  static void set_abort(Fl_Abort_Handler f) {fatal = f;}
1221  static void (*atclose)(Fl_Window*, void*);
1225  static void default_atclose(Fl_Window *);
1226  static void set_atclose(Fl_Atclose_Handler f) {atclose = f;}
1228  static int event_shift() {return e_state&FL_SHIFT;}
1229  static int event_ctrl() {return e_state&FL_CTRL;}
1230  static int event_command() {return e_state&FL_COMMAND;}
1231  static int event_alt() {return e_state&FL_ALT;}
1232  static int box_dx(Fl_Boxtype);
1233  static int box_dy(Fl_Boxtype);
1234  static int box_dw(Fl_Boxtype);
1235  static int box_dh(Fl_Boxtype);
1237  static void grab(Fl_Window* win) {grab(win);}
1238  static void release() {grab(0);}
1239
1240  // Visible focus methods...
1244  static void visible_focus(int f) {option(OPTION_VISIBLE_FOCUS, (f!=0));}
1245  static int visible_focus() { return option(OPTION_VISIBLE_FOCUS); }
1249  // Drag-n-drag text operation methods...
1254  static void dnd_text_ops(int f) {option(OPTION_DND_TEXT, (f!=0));}
1255  static int dnd_text_ops() { return option(OPTION_DND_TEXT); }
1258  // Multithreading support:
1263  static int lock();
1264  static void unlock();
1265
1269  // Visible focus methods...
1274  static void use_high_res_GL(int val) { use_high_res_GL_ = val; }
1275  static int use_high_res_GL() { return use_high_res_GL_; }
1277
1280  // Cairo support API
1284  static cairo_t *cairo_make_current(Fl_Window *); 
1285  static void cairo_autolink_context(bool alink) {cairo_state_.autolink(alink);}
1286  static bool cairo_autolink_context() { return cairo_state_.autolink();}
1287
1290  // Convert Windows commandline arguments to UTF-8 (documented in src/Fl.cxx)
1294  static int args_to_utf8(int argc, char ** &argv);
static void cairo_flush(cairo_t *c) {
  // flush Cairo drawings: necessary at least for Windows
  cairo_surface_t *s = cairo_get_target(Fl::cairo_cc());
  cairo_surface_flush(s);
}

private:

static cairo_t *cairo_make_current(void *gc);
static cairo_t *cairo_make_current(void *gc, int W, int H);
static Fl_Cairo_State cairo_state_;
13.8 Fl_Anim_GIF_Image.H

```c
#ifndef Fl_Anim_GIF_Image_H
#define Fl_Anim_GIF_Image_H

// forward declarations
class Fl_Image;
class Fl_Widget;

#include <FL/Fl_GIF_Image.H>

// Load and display animated GIF images
class FL_EXPORT Fl_Anim_GIF_Image : public Fl_GIF_Image {

  class FrameInfo; // internal helper class

public:

  enum Flags {
    DONT_START = 1,
    DONT_RESIZE_CANVAS = 2,
    DONT_SET_AS_IMAGE = 4,
    OPTIMIZE_MEMORY = 8,
    LOG_FLAG = 64,
    DEBUG_FLAG = 128
  };

  // -- constructors and destructor
  Fl_Anim_GIF_Image(const char *filename, Fl_Widget *canvas = 0, unsigned short flags = 0);
  Fl_Anim_GIF_Image(const char * imagename, const unsigned char *data, const size_t length, Fl_Widget *canvas = 0,
    unsigned short flags = 0);
  Fl_Anim_GIF_Image();
  ~Fl_Anim_GIF_Image() FL_OVERRIDE;

  // -- file handling
  bool load(const char *name, const unsigned char *imgdata=NULL, size_t imglength=0);
  bool valid() const;

  // -- getters and setters
  void frame_uncache(bool uncache);
  bool frame_uncache() const;
  double delay(int frame_) const;
  void delay(int frame, double delay);
  void canvas(Fl_Widget *canvas, unsigned short flags = 0);
  Fl_Widget *canvas() const;
  int canvas_w() const;
  int canvas_h() const;
  bool is_animated() const;
  const char *name() const;
  void speed(double speed);
  double speed() const;

  // -- animation
  int frames() const;
  void frame(int frame);
  int frame() const;
  Fl_Image *image() const;
  Fl_Image *image(int frame) const;
  bool start();
  bool stop();
  bool next();

  bool playing() const { return valid() && Fl::has_timeout(cb_animate, (void *)this); }

  // -- image data
  bool playing() const { return valid() && Fl::has_timeout(cb_animate, (void *)this); }

  Fl_Anim_GIF_Image& resize(int w, int h);
  Fl_Anim_GIF_Image& resize(double scale);
  int frame_x(int frame) const;
  int frame_y(int frame) const;
  int frame_w(int frame) const;

};
#endif // Fl_Anim_GIF_Image_H
```

Generated by Doxygen
```cpp
int frame_h(int frame) const;

// -- overridden methods
void color_average(Fl_Color c, float i) FL_OVERRIDE;
Fl_Image *copy(int W, int H) const FL_OVERRIDE;
void de saturate() FL_OVERRIDE;
draw(int x, int y, int w, int h, int cx = 0, int cy = 0) FL_OVERRIDE;
void uncache() FL_OVERRIDE;

// -- debugging and logging
int debug() const;

// -- static methods
int frame_count(const char *name, const unsigned char *imgdata = NULL, size_t imglength = 0);

protected:
bool next_frame();
void clear_frames();
void set_frame(int frame);

static void cb_animate(void *d);
scale_frame();
set_frame();
on_frame_data(Fl_GIF_Image::GIF_FRAME &f) FL_OVERRIDE;
on_extension_data(Fl_GIF_Image::GIF_FRAME &f) FL_OVERRIDE;

private:
char *name_;
unsigned short flags_;
Fl_Widget *canvas_;
bool uncache_;
bool valid_; int frame_; double speed_; FrameInfo *fi_;

#endif // Fl_Anim_Gif_Image_H
```

---

### 13.9 fl_ask.H File Reference

API for common dialogs.

```
#include <FL/Enumerations.H>
#include <FL/fl_attr.h>
```

#### Enumerations

```
enum Fl_Beep {
    FL_BEEP_DEFAULT = 0, FL_BEEP_MESSAGE, FL_BEEP_ERROR, FL_BEEP_QUESTION,
    FL_BEEP_PASSWORD, FL_BEEP_NOTIFICATION
}
```

Defines the different system beeps available.

#### Functions

```
void void fl_alert (const char *,...) __fl_attr((__format__(__printf__

void void fl_input (const char *,const char *,...) __fl_attr((__format__(__printf__

void fl_beep (int type=FL_BEEP_DEFAULT)
```

Emits a system beep.

```
int fl_choice (const char *,...,) __fl_attr((__format__(__printf__

int const char const char *fl_choice_n (const char *,...,) __fl_attr((__format__(__printf__

int const char const char const char *fl_input (const char *,...,) __fl_attr((__format__(__printf__
```
13.9 fl_ask.H File Reference 2011

- int const char const char * fl_input (int maxchar, const char *label, const char *deflt=0,...) __fl_attr(__format__(__printf__)
- void fl_message (const char *,...) __fl_attr(__format__(__printf__)
- void fl_message_font (Fl_Font f, Fl_Fontsize s)
- void fl_message_hotspot (int enable)
  Sets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.
- int fl_message_hotspot (void)
  Gets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.
- int const char const char int const char const char Fl_Widget * fl_message_icon ()
  Gets the Fl_Box icon container of the current default dialog used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_icon_label (const char *str)
  Sets the icon label of the dialog window used in many common dialogs.
- void fl_message_position (const int x, const int y, const int center=0)
  Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_position (Fl_Widget &widget)
- void fl_message_position (Fl_Widget *widget)
  Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- int fl_message_position (int *x=0, int *y=0)
  Gets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_title (const char *title)
  Sets the title of the dialog window used in many common dialogs.
- void fl_message_title_default (const char *title)
  Sets the default title of the dialog window used in many common dialogs.
- int const char const char * fl_password (const char *label, const char *deflt=0,...) __fl_attr(__format__(__printf__)
- int const char const char int const char const char * fl_password (int maxchar, const char *label, const char *deflt=0,...) __fl_attr(__format__(__printf__

Variables

- void void int __deprecated__
- const char * fl_cancel
  string pointer used in common dialogs, you can change it to another language
- const char * fl_close
  string pointer used in common dialogs, you can change it to another language
- Fl_Font fl_message_font_
- Fl_Fontsize fl_message_size_
- const char * fl_no
  string pointer used in common dialogs, you can change it to another language
- const char * fl_ok
  string pointer used in common dialogs, you can change it to another language
- const char * fl_yes
  string pointer used in common dialogs, you can change it to another language

13.9.1 Detailed Description

API for common dialogs.
13.9.2 Enumeration Type Documentation

13.9.2.1 Fl_Beep

```c
enum Fl_Beep
```

Defines the different system beeps available. Some systems may play different sounds or use different sound volume depending on the Fl_Beep value. The implementation is platform dependent.

See also

`fl_beep(int)`

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_BEEP_DEFAULT</td>
<td>Default beep.</td>
</tr>
<tr>
<td>FL_BEEP_MESSAGE</td>
<td>Message beep.</td>
</tr>
<tr>
<td>FL_BEEP_ERROR</td>
<td>Error beep.</td>
</tr>
<tr>
<td>FL_BEEP_QUESTION</td>
<td>Question beep.</td>
</tr>
<tr>
<td>FL_BEEP_PASSWORD</td>
<td>Password beep.</td>
</tr>
<tr>
<td>FL_BEEP_NOTIFICATION</td>
<td>Notification beep.</td>
</tr>
</tbody>
</table>

13.9.3 Function Documentation

13.9.3.1 fl_message_position()

```c
void fl_message_position ( Fl_Widget & widget ) [inline]
```

See also

`fl_message_position(Fl_Widget *widget)`. 

13.10 fl_ask.H

Go to the documentation of this file.
FL_BEEP_QUESTION,
FL_BEEP_PASSWORD,
FL_BEEP_NOTIFICATION
};

FL_EXPORT void fl_beep(int type = FL_BEEP_DEFAULT);
FL_EXPORT void fl_message(const char *s, ...) __fl_attr((__format__(__printf__, 1, 2))); // Use fl_choice() or fl_choice_n() with the appropriate verbs instead.

FL_EXPORT void fl_alert(const char *s, ...) __fl_attr((__format__(__printf__, 1, 2)));

FL_EXPORT int fl_ask(const char *s, ...) __fl_attr((__format__(__printf__, 1, 2), __deprecated__));

FL_EXPORT int fl_choice(const char *q, const char *b0, const char *b1, const char *b2, ...)
__fl_attr((__format__(__printf__, 1, 5))); // since FLTK 1.3.8:

FL_EXPORT const char *fl_input(const char *label, const char *deflt = 0, ...)
__fl_attr((__format__(__printf__, 1, 3))); // with extended return value (-2, -1, 0, 1, 2)

FL_EXPORT const char *fl_password(const char *label, const char *deflt = 0, ...)
__fl_attr((__format__(__printf__, 1, 3))); // since FLTK 1.4.0:

FL_EXPORT int fl_choice_n(const char *q, const char *b0, const char *b1, const char *b2, ...)
__fl_attr((__format__(__printf__, 1, 5))); // since FLTK 1.4.0:

FL_EXPORT const char *fl_input(int maxchar, const char *label, const char *deflt = 0, ...)
__fl_attr((__format__(__printf__, 2, 4))); // since FLTK 1.4.0:

FL_EXPORT const char *fl_password(int maxchar, const char *label, const char *deflt = 0, ...)
__fl_attr((__format__(__printf__, 2, 4))); // since FLTK 1.4.0:

# ifdef (FLTK_USE_STD)

FL_EXPORT std::string fl_input_str(int maxchar, const char *label, const char *deflt = 0, ...)
__fl_attr((__format__(__printf__, 2, 4))); // since FLTK 1.4.0:

FL_EXPORT std::string fl_input_str(int &ret, int maxchar, const char *label, const char *deflt = 0, ...)
__fl_attr((__format__(__printf__, 3, 5))); // since FLTK 1.4.0:

FL_EXPORT std::string fl_password_str(int maxchar, const char *label, const char *deflt = 0, ...)
__fl_attr((__format__(__printf__, 2, 4))); // since FLTK 1.4.0:

FL_EXPORT std::string fl_password_str(int &ret, int maxchar, const char *label, const char *deflt = 0, ...)
__fl_attr((__format__(__printf__, 3, 5))); // since FLTK 1.4.0:

# endif

FL_EXPORT Fl_Widget *fl_message_icon();
extern FL_EXPORT Fl_Font fl_message_font_;
extern FL_EXPORT Fl_Fontsize fl_message_size_;
inline void fl_message_font(Fl_Font f, Fl_Fontsize s) {
  fl_message_font_ = f;
  fl_message_size_ = s;
}

FL_EXPORT void fl_message_hotspot(int enable);
FL_EXPORT int fl_message_hotspot(void);

// pointers you can use to change FLTK to another language:

Generated by Doxygen
This file defines compiler-specific macros.

Macros

- `#define __fl_attr(x)`
  This section lists macros for Doxygen documentation only.

- `#define FL_DEPRECATED(msg, func)`
  Enclosing a function or method in `FL_DEPRECATED` marks it as no longer recommended.

- `#define FL_OVERRIDE` override
  This macro makes it safe to use the C++11 keyword `override` with older compilers.

### 13.11.1 Detailed Description

This file defines compiler-specific macros.

#### 13.11.2 Macro Definition Documentation

**13.11.2.1 __fl_attr**

```
#define __fl_attr( 
    x )
```

This section lists macros for Doxygen documentation only.

The next section will define the actual macros based on the compile used and based on the capabilities of the version of that compiler. To be used in prototypes with a variable list of arguments. This macro helps detection of mismatches between format string and argument list at compilation time.

Usage example: FL/fl_ask.H

**13.11.2.2 FL_DEPRECATED**

```
#define FL_DEPRECATED( 
    msg, 
    func )
```

Value:

```
/**/ \deprecated msg \*/ 
func
```

Enclosing a function or method in `FL_DEPRECATED` marks it as no longer recommended.

This macro syntax can not be used if the return type contains a comma, which is not the case in FLTK.

```
FL_DEPRECATED("Outdated, don't use", int position()) { return position_; }
```

### 13.12 fl_attr.h

Go to the documentation of this file.
/*

Declare macros specific to Visual Studio.

Visual Studio defines __cplusplus = '199711L' in all its versions which is not helpful for us here. For VS version number encoding see:

This document specifies that the macro _MSVC_LANG is defined since "Visual Studio 2015 Update 3" as 201402L (default) and undefined in earlier versions. It can be used to determine the C++ standard as specified by the /std:c++ compiler option:
- /std:c++14 201402L (also if /std:c++ is not used)
- /std:c++17 201703L
- /std:c++20 202002L
- /std:c++latest a "higher, unspecified value" (docs of VS 2022)

As of this writing (02/2023) _MSVC_LANG is not yet used in this file but it is documented for future use.

*/

#if defined(_MSC_VER)
// Visual Studio 2005 (8.0)
#define FL_DEPRECATED(msg, func) __declspec(deprecated(msg)) func
#endif // Visual Studio .NET 2003 (7.1)

#endif // Visual Studio .NET 2003 (7.1)
#endif // Visual Studio

// Declare macros specific to the C++ standard used.
#define FL_OVERRIDE override

// Macros may have been declared already in previous sections.
*/
#define (__cplusplus >= 202002L) // C++20
#define (__cplusplus >= 201703L) // C++17
#define (__cplusplus >= 201402L) // C++14
#define (__cplusplus >= 201103L) // C++11
#define(__cplusplus >= 200000L) // C++0x
#define(_MSC_VER >= 1900) // Visual Studio 2015 (14.0)
#define(_MSC_VER >= 1700) // Visual Studio 2017 (15.0)
#define(_MSC_VER >= 1400) // Visual Studio 2005 (8.0)
#define(_MSC_VER >= 1310) // Visual Studio .NET 2003 (7.1)
#define(_MSC_VER >= 1202) // Visual Studio .NET 2002 (6.0)
#define(_MSC_VER >= 1000) // Visual Studio 2000 (6.0)
#define(_MSC_VER >= 9800) // Visual Studio 2000 (6.0)

#ifndef __fl_attr(x)
#endif // __fl_attr(x)

#endif // FL_DOXYGEN */
#endif // _FL_FL_ATTR_H_
#endif // _FL_DOXYGEN
#endif // __cplusplus
#endif // FL_DOXYGEN

void foo();

__fl_attr(x)

foo();
#ifndef __fl_attr
#define __fl_attr(x)
#endif

#ifndef FL_OVERRIDE
#define FL_OVERRIDE
#endif

#ifndef FL_DEPRECATED
#define FL_DEPRECATED(msg, func) func
#endif

#endif /* _FL_fl_attr_h_ */

13.13 Fl_Bitmap.H

// Bitmap header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2017 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php

Generated by Doxygen
13.14 Fl_BMP_Image.H

00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 // https://www.fltk.org/bugs.php
00015 //
00016 //
00017 /* \file
00018 Fl_Bitmap widget . */
00019
00020 ifndef Fl_Bitmap_H
00021 define Fl_Bitmap_H
00022 #include "Fl_Image.H"
00023 #include "Fl_Widget.H" // for fl_uintptr_t
00024
00025 class Fl_Widget;
00026 struct Fl_Menu_Item;
00027
class FL_EXPORT Fl_Bitmap : public Fl_Image {
00030 friend class Fl_Graphics_Driver;
00033 public:
00036 const uchar *array;
00038 int alloc_array;
00040 private:
00042 fl_uintptr_t id_; // size of bitmap when cached
00043
00045 public:
00048 Fl_Bitmap(const uchar *bits, int W, int H) :
00050 Fl_Image(W,H,0), array(bits), alloc_array(0), id_(0), cache_w_(0), cache_h_(0) {data((const char ***)&array, 1);}
00052 Fl_Bitmap(const char *bits, int W, int H) :
00053 Fl_Image(W,H,0), array((const uchar *)bits), alloc_array(0), id_(0), cache_w_(0), cache_h_(0)
00055 {data((const char ***)&array, 1);}
00056 Fl_Bitmap(const uchar *bits, int bits_length, int W, int H);
00057 Fl_Bitmap(const char *bits, int bits_length, int W, int H);
00058 virtual ~Fl_Bitmap();
00059 FIImage *copy() const { return FIImage::copy(); }
00060 void draw(int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE;
00061 void label(Fl_Widget *w) FL_OVERRIDE;
00062 void uncache() FL_OVERRIDE;
00064 int cache_w() { return cache_w_;}
00065 int cache_h() { return cache_h_;}
00066 };
00067
00068 endif

13.14 Fl_BMP_Image.H

00001 //
00002 // BMP image header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2022 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file. If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 // https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 // https://www.fltk.org/bugs.php
00015 //
00016 //
00017 /* \file
00018 Fl_BMP_Image widget . */
00019
00020 ifndef Fl_BMP_Image_H
00021 define Fl_BMP_Image_H
00022 #include "Fl_Image.H"
00023
class FL_EXPORT Fl_BMP_Image : public Fl_RGB_Image {
00030 public:
00032 Fl_BMP_Image(const char *filename);
00033 Fl_BMP_Image(const char *imagename, const unsigned char *data, const long length = -1);
00036 protected:
00038 void load_bmp_(class Fl_Image_Reader &rdr, int ico_height = 0, int ico_width = 0);
13.15 Fl_Box.H File Reference

Fl_Box widget.

```c
#include "Fl_Widget.H"
```

Classes

- class Fl_Box

  This widget simply draws its box, and possibly its label.

13.16 Fl_Box.H

Go to the documentation of this file.

```c
// Box header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2023 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#ifndef Fl_Box_H
#define Fl_Box_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

class FL_EXPORT Fl_Box : public Fl_Widget {

protected:
  void draw() FL_OVERRIDE;

public:
  Fl_Box(int X, int Y, int W, int H, const char *l=0);
  Fl_Box(Fl_Boxtype b, int X, int Y, int W, int H, const char *l);
  int handle(int) FL_OVERRIDE;
};
#endif
```

13.17 Fl_Browser.H

```c
// Browser header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2023 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
```
/* Form-compatible browser. Probably useful for other lists of textual data. Notice that the line numbers start from 1, and 0 means "no line".

#ifndef Fl_Browser_H
#define Fl_Browser_H

#include "Fl_Browser_.H"
#include "Fl_Image.H"

struct FL_BLINE;

class FL_EXPORT Fl_Browser : public Fl_Browser_ {

    FL_BLINE *first; // the array of lines
    FL_BLINE *last;
    FL_BLINE *cache;
    int cacheline; // line number of cache
    int lines; // Number of lines
    int full_height_; // alternative to $-sign
    char format_char_; // alternative to @-sign
    char column_char_; // alternative to tab

protected:
    FL_BLINE * find_line(int line) const ;
    FL_BLINE * _remove(int line) ;
    void insert(int line, FL_BLINE * item);
    int lineno(void *item) const ;
    void swap(FL_BLINE *a, FL_BLINE *b);

public:
    void remove(int line);
    void add(const char * newtext, void* d = 0);
    void insert(int line, const char * newtext, void* d = 0);
    void move(int to, int from);
    int load(const char * filename);
    void swap(int a, int b);
    void clear();

int size() const { return lines; }

size(int W, int H) { Fl_Widget::size(W, H); }

int textsize() const { return Fl_Browser_::textsize(); }

void textsize(Fl_Fontsize newSize);

int topline() const ;
enum Fl_Line_Position { TOP, BOTTOM, MIDDLE };
void lineposition(int line, Fl_Line_Position pos);
void topline(int line) { lineposition(line, TOP); }
void bottomline(int line) { lineposition(line, BOTTOM); }
void middileline(int line) { lineposition(line, MIDDLE); }

int select(int line, int val=1);
int selected(int line) const ;
void show(int line);
void show() FL_OVERRIDE { Fl_Widget::show(); }
void hide(int line);
*/
void hide() FL_OVERRIDE { Fl_Widget::hide(); }  
int visible(int line) const ;  
int value() const ;  
void value(int line) { select(line); }  
const char * text(int line) const ;  
void text(int line, const char * newtext);  
void* data(int line) const ;  
void data(int line, void * d);  

Fl_Browser(int X, int Y, int W, int H, const char *L = 0);  
~Fl_Browser() { clear(); }  

char format_char() const { return format_char_; }  
void format_char(char c) { format_char_ = c; }  
char column_char() const { return column_char_; }  
void column_char(char c) { column_char_ = c; }  
const int * column_widths() const { return column_widths_; }  
void column_widths(const int * arr) { column_widths_ = arr; }  

int displayed(int line) const { return Fl_Browser_::displayed(find_line(line)); }  

void make_visible(int line) {  
if (line < 1) Fl_Browser_::display(find_line(1));  
else if (line > lines) Fl_Browser_::display(find_line(lines));  
else Fl_Browser_::display(find_line(line));  
}  

// icon support  
void icon(int line, Fl_Image * icon);  
Fl_Image * icon(int line) const;  
void remove_icon(int line);  

void replace(int a, const char * b) { text(a, b); }  
void display(int line, int val=1);  

Fl_Browser_.H

13.18 Fl_Browser_.H

// Common browser header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2016 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php
// Yes, I know this should be a template...

 ifndef Fl_Browser_.H
 define Fl_Browser_.H
 ifndef Fl_Group.H
 include "Fl_Group.H"
 endif
 include "FL_Scrollbar.H"  
 include <FL/Fl.H>  
 define FL_NORMAL_BROWSER 0  
 define FL_SELECT_BROWSER 1  
 define FL_HOLD_BROWSER 2  
 define FL_MULTI_BROWSER 3  
 define FL_SORT_ASCENDING 0  
 define FL_SORT_DESCENDING 1  
 define FL_SORT_CASESENSITIVE 0x2
 class Fl_EXPORT Fl_Browser_ : public Fl_Group {  
 int position_;  // where user wants it scrolled to  
 int real_position_;  // the current vertical scrolling position  
 int hposition_;  // where user wants it panned to  
 int real_hposition_;  // the current horizontal scrolling position  
 int offset_;  // how far down top_ item the real_position is  
 int max_width;  // widest object seen so far

uchar has_scrollbar_; // which scrollbars are enabled
FL_Font textfont_;  
FL_Fontsize textsize_;  
FL_Color textcolor_;  
void* top_; // which item scrolling position is in 
void* selection_; // which is selected (except for FL_MULTI_BROWSER)  
void* max_width_item; // which item has max_width_  
intscrollbar_size_; // size of scrollbar trough 
linespacing_; 
protected: 
// All of the following must be supplied by the subclass: 
virtual void *item_first() const = 0;  
virtual void *item_next(void *item) const = 0;  
virtual void *item_prev(void *item) const = 0;  
virtual void *item_last() const { return 0L; }  
virtual int item_height(void *item) const = 0;  
virtual int item_width(void *item) const = 0;  
virtual int item_quick_height(void *item) const;  
virtual void *item_draw(void *item,int X,int Y,int W,int H) const = 0;  
virtual const char *item_text(void *item) const { (void)item; return 0L; }  
virtual void item_swap(void *a,void *b) { (void)a; (void)b; }  
virtual int item_selected(void *item) const;  
virtual int full_width() const ; // current width of all items  
virtual int full_height() const ; // current height of all items  
virtual int incr_height() const ; // average height of an item  
virtual int select(void *item,int val=1);  
virtual int select_only(void *item,int docallbacks=0);  
virtual int deselect(int docallbacks=0);  
int handle(int event) FL_OVERRIDE;  
int hposition() const { return hposition_; }  
void hposition(int); // pan to here  
void display(void *item); // scroll so this item is shown

public:  
public: 
void draw() FL_OVERRIDE; 
void new_list(); // completely clobber all data, as though list replaced
void deleting(void *item); // get rid of any pointers to item 
void regen(void *a,void *b); // change a pointers to b  
void swapping(void *a,void *b); // exchange pointers a and b  
void vposition(int pos); // scroll to here
```cpp
void has_scrollbar(uchar mode) { has_scrollbar_ = mode; }

Fl_Font textfont() const { return textfont_; }
void textfont(Fl_Font font) { textfont_ = font; }

Fl_Fontsize textsize() const { return textsize_; }
void textsize(Fl_Fontsize newSize) { textsize_ = newSize; }

Fl_Color textcolor() const { return textcolor_; }
void textcolor(Fl_Color col) { textcolor_ = col; }

int scrollbar_size() const {
    return(scrollbar_size_);
}
void scrollbar_size(int newSize) {
    scrollbar_size_ = newSize;
}

int scrollbar_width() const {
    return(Fl::scrollbar_size());
}
void scrollbar_width(int width) {
    Fl::scrollbar_size(width);
    scrollbar_size_ = 0;
}

void scrollbar_right() { scrollbar.align(FL_ALIGN_RIGHT); }
void scrollbar_left() { scrollbar.align(FLALIGN_LEFT); }

void sort(int flags=0);

void linespacing(int pixels) { linespacing_ = pixels; }

int linespacing() const { return linespacing_; }

#endif
```

13.19 Fl_Button.H

```
/*
file Fl_Button widget . */

#ifndef Fl_Button_H
#define Fl_Button_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

// values for type()
define FL_NORMAL_BUTTON 0
#define FL_TOGGLE_BUTTON 1
#define FL_RADIO_BUTTON (FL_RESERVED_TYPE+2)
#define FL_HIDDEN_BUTTON 3

extern FL_EXPORT Fl_Shortcut fl_old_shortcut(const char*);

class Fl_Widget_Tracker;

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#ifndef Fl_Widget_Tracker;
#endif
#endif

#define FL_NORMAL_BUTTON 0
#define FL_TOGGLE_BUTTON 1
#define FL_RADIO_BUTTON (FL_RESERVED_TYPE+2)
#define FL_HIDDEN_BUTTON 3

extern FL_EXPORT Fl_Shortcut fl_old_shortcut(const char*);

class Fl_Widget_Tracker;

class_EXPORT Fl_Button fl_old_shortcut(const char*);

class Fl_Widget_Tracker;

class EXPORT Fl_Button : public Fl_Widget {
        
int shortcut_; 
char value_; 
uchar down_box_; 
uchar compact_; 

protected: 
static Fl_Widget_Tracker *key_release_tracker;
static void key_release_timeout(void*);
```
void simulate_key_action();
void draw() FL_OVERRIDE;

public:
int handle(int) FL_OVERRIDE;

Fl_Button(int X, int Y, int W, int H, const char *L = 0);

int value(int v);
char value() const {return value_);} int set() {return value(1);} int clear() {return value(0);} void setonly(); // this should only be called on FL_RADIO_BUTTONS
int shortcut() const {return shortcut_;}
void shortcut(int s) {shortcut_ = s;}
void shortcut(const char *s) {shortcut(fl_old_shortcut(s));}

Fl_Boxtype down_box() const {return (Fl_Boxtype)down_box_;} void down_box(Fl_Boxtype b) {down_box_ = b;}

Fl_Color down_color() const {return selection_color();}
void down_color(unsigned c) {selection_color(c);}

// handle flag for compact buttons, documentation in source code
void compact(uchar v);
uchar compact() { return compact_; }

#endif

Cairo is currently supported for the following platforms: Windows, macOS, Unix/Linux (X11 + Wayland).
#include <FL/Fl.H>
#include <cairo.h>

Classes

• class Fl_Cairo_State

Contains all the necessary info on the current cairo context.

Detailed Description

Cairo is currently supported for the following platforms: Windows, macOS, Unix/Linux (X11 + Wayland).

Note

In FLTK 1.3.x this header file (Fl_Cairo.H) included the platform specific Cairo headers. This is no longer true since 1.4.0.

This header file is platform agnostic. If you need platform specific Cairo headers you need to #include them in your source file.

To use FLTK's builtin Cairo support you need to #include <FL/Fl.H> before you include any other FLTK header which is officially required anyway. Since FLTK 1.4.0 the preprocessor constants FLTK_HAVE_CAIRO and/or FLTKHAVE_CAIROEXT are defined in <FL/Fl.H> by including <FL/fl_config.h>.

Go to the documentation of this file.
Main Cairo support header file for the Fast Light Tool Kit (FLTK).

Copyright 1998-2023 by Bill Spitzak and others.

This library is free software. Distribution and use rights are outlined in the file "COPYING" which should have been included with this file. If this file is missing or damaged, see the license at:

https://www.fltk.org/COPYING.php

Please see the following page on how to report bugs and issues:

https://www.fltk.org/bugs.php

ifndef FL_CAIRO_H
#define FL_CAIRO_H

#include <FL/Fl.H>

ifdef FLTK_HAVE_CAIRO

#include <cairo.h>

class FL_EXPORT Fl_Cairo_State {
public:
  Fl_Cairo_State()
  : cc_(0)
  , own_cc_(false)
  , autolink_(false)
  , window_(0)
  , gc_(0) {}

  // access attributes
  cairo_t *cc() const { return cc_; } // return the current cairo context
  bool autolink() const { return autolink_; } // true if autolink is enabled
  void cc(cairo_t *c, bool own = true) { // set the cairo context
    if (cc_ && own_cc_)
      cairo_destroy(cc_);
    cc_ = c;
    if (!cc_)
      window_ = 0;
    own_cc_ = own;
  }
  void autolink(bool b);
  void window(void *w) { window_ = w; }
  void gc(void *c) { gc_ = c; }

private:
  cairo_t *cc_; // contains the unique autoupdated cairo context
  bool own_cc_; // indicates whether we must delete the cc, useful for internal cleanup
  bool autolink_; // false by default, prevents the automatic cairo mapping on fltk windows
  void *window_, *gc_; // for keeping track internally of last win+gc treated
#endif // FLTK_HAVE_CAIRO
#endif // FL_CAIRO_H

13.22 Fl_Cairo_Window.H File Reference

Fl_Cairo_Window, an FLTK window incorporating a Cairo draw callback.

#include <FL/fl_config.h>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>

Classes

- class Fl_Cairo_Window

  This defines an FLTK window with Cairo support.

13.22.1 Detailed Description

Fl_Cairo_Window, an FLTK window incorporating a Cairo draw callback.
13.23 Fl_Cairo_Window.H

Go to the documentation of this file.

#include <FL/fl_config.h>

#ifdef FLTK_HAVE_CAIRO

// Cairo is currently supported for the following platforms:
// Win32, Apple Quartz, X11, Wayland

#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>

class FL_EXPORT Fl_Cairo_Window : public Fl_Double_Window {

public:
    Fl_Cairo_Window(int W, int H, const char *L = 0)
    : Fl_Double_Window(W, H, L), draw_cb_(0) {}
    Fl_Cairo_Window(int X, int Y, int W, int H, const char *L = 0)
    : Fl_Double_Window(X, Y, W, H, L), draw_cb_(0) {}

protected:
    void draw() FL_OVERRIDE {
        Fl_Double_Window::draw();
        if (draw_cb_) { // call the Cairo draw callback
            Fl::cairo_make_current(this);
            draw_cb_(this, Fl::cairo_cc());
            Fl::cairo_flush(Fl::cairo_cc());
        }
    }

public:
    typedef void ( *cairo_draw_cb) (Fl_Cairo_Window* self, cairo_t* def);
    void set_draw_cb(cairo_draw_cb cb) { draw_cb_ = cb; }

private:
    cairo_draw_cb draw_cb_;


Creates code to declare a callback function in line with instantiating a widget.

- \#define FL_METHOD_CALLBACK_1(WIDGET, CLASS, SELF, METH, TYPE0, VALUE0)

Declares a non-static class method callback with custom parameters.

13.24.1 Detailed Description

This file provides macros for easy function and method callbacks with multiple type-safe arguments.

13.24.2 Macro Definition Documentation

13.24.2.1 FL_FUNCTION_CALLBACK_3

\#define FL_FUNCTION_CALLBACK_3(
  WIDGET,
  FUNC,
  TYPE0,
  VALUE0,
  TYPE1,
  VALUE1,
  TYPE2,
  VALUE2
)

Declares a C function callback with custom parameters.

You can declare a plain C function callback or a static method callback with custom parameters using this macro. It simplifies the process of calling arbitrary functions with up to five custom parameters. The macro generates code that ensures type safety and expands FLTK’s standard callbacks, which are limited to a single `void*` or `long` argument.

To use the macro, you provide the widget that will handle the callback as the first argument. The second argument can be either a regular function or a static method in any class.

Following these arguments, you can include up to five pairs, where each pair consists of a type and a value. For example, `int, 3` specifies an integer parameter with a value of 3. If you need to pass two arguments, you can use two pairs, like this: `int, 3, int, 4`. The last digit of the macro name must be the same as the number of pairs (0..5)

Whenever the code generated by the macro is called, the custom parameters are duplicated and marked for automatic deallocation using `delete` when the callback widget is destroyed.

```
#include <FL/fl_callback_macros.H>
...
Fl_Button *btn1 = new Fl_Button(10, 10, 100, 20, "Beep");
FL_FUNCTION_CALLBACK_0(btn1, fl_beep);
...
Fl_Button *btn2 = new Fl_Button(10, 40, 100, 20, "Hello");
FL_FUNCTION_CALLBACK_5(btn2,
  fl_message,
  const char *, text, "Hello\n%d %d %d %d",
  int, a, 1, int, b, 2, int, c, 3, int, d, 4
);
```

You can find a small demonstration program showcasing the usage of `FL_*CALLBACK_*` in the `examples/callbacks.cxx` file.

Parameters

<table>
<thead>
<tr>
<th>WIDGET</th>
<th>the widget that will call the callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNC</td>
<td>a C/C++ function or a static class method</td>
</tr>
<tr>
<td>TYPE0,VALUE0,TYPE1,VALUE1,TYPE2,VALUE2</td>
<td>a list of zero to five type/value pairs, all separated by commas</td>
</tr>
</tbody>
</table>

See also

`FL_METHOD_CALLBACK_1, FL_INLINE_CALLBACK_2`

13.24.2.2 FL_INLINE_CALLBACK_2

\#define FL_INLINE_CALLBACK_2(
  WIDGET,
  ...
)
Creates code to declare a callback function in line with instantiating a widget.

You can use this macro to create a function as a callback, allowing you to define the callback function right where the widget and callback are declared, similar to a Lambda function.

The first argument of the macro specifies the widget that will handle the callback. Next, you can include up to five triplets, where each triplet consists of a type, a parameter name, and a value. For example, `int, x, 3` specifies an integer parameter with a value of 3. If you need to pass two arguments, you can use two triplets, such as `int, x, 3, int, y, 4`. The last digit of the macro name must be the same as the number of triplets (0..5).

The last argument is the actual function body itself.

The function body is limited to a syntax that the macro preprocessor can handle. It should include the leading '{' and trailing '}' and may contain local variable declarations, use global variables and functions, and use also the variables listed and initialized in the argument triples of the macro. Very large function bodies should be avoided because they may exceed the admissible size of a macro argument.

Whenever the code generated by the macro is called, the custom parameters are duplicated and marked for automatic deallocation using `delete` when the callback widget is destroyed.

```c++
#include <FL/fl_callback_macros.H>
...
Fl_Button *btn = new Fl_Button(10, 10, 100, 20, "Test");
FL_INLINE_CALLBACK_1(btn,
    const char *, name, btn->label(),
    { fl_message("Greetings from the %s button", name); }
);
```

You can find a small demonstration program showcasing the usage of `FL_*CALBACK_*` in the examples/callbacks.cxx file.

**Parameters**

<table>
<thead>
<tr>
<th><strong>Parameter</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>WIDGET</td>
<td>the widget that will call the callback</td>
</tr>
<tr>
<td>TYPE0</td>
<td>the type of the first parameter in the function call</td>
</tr>
<tr>
<td>NAME0</td>
<td>an arbitrary variable name that can be used as a parameter in the function body</td>
</tr>
<tr>
<td>VALUE0</td>
<td>a constant value or a variable; the value of the variable is copied when the callback is created</td>
</tr>
<tr>
<td>TYPE1, NAME1, VALUE1</td>
<td>as above; there are six macros that support 0 to 5 parameters</td>
</tr>
<tr>
<td>LAMBDA</td>
<td>the function body within the limits of the C macro preprocessor</td>
</tr>
</tbody>
</table>

**See also**

`FL_METHOD_CALLBACK_1, FL_FUNCTION_CALLBACK_3`

### 13.24.2.3 FL_METHOD_CALLBACK_1

```c++
#define FL_METHOD_CALLBACK_1(
    WIDGET,
    CLASS,
    SELF,
    METH,
    TYPE0,
    VALUE0 )
```

Declare a non-static class method callback with custom parameters.

You can declare a callback for a non-static class method with custom parameters using this macro. It provides a convenient way to call arbitrary methods in any class, overcoming FLTK's limitation of passing only a single `void*` or `long` argument. Furthermore, it ensures type safety.
The first argument of the macro specifies the widget that will handle the callback. The second argument indicates the class type to be called. The third argument must be a pointer to an instance of that class. The fourth argument is the name of the method within the class. That method must be public and should not be static.

Following these arguments, you can include up to five pairs, where each pair consists of a type and a value. For example, `int, 3` specifies an integer parameter with a value of 3. If you need to pass two arguments, you can use two pairs, like this: `int, 3, int, 4`. The last digit of the macro name must be the same as the number of pairs (0..5).

Whenever the code generated by the macro is called, the custom parameters are duplicated and marked for automatic deallocation using `delete` when the callback widget is destroyed.

```c
#include <FL/fl_callback_macros.H>

FL_Button *btn = new FL_Button(10, 10, 100, 20, "Test");
FL_METHOD_CALLBACK_1(btn, Fl_Button, self, color, Fl_Color, FL_GREEN);
```

You can find a small demonstration program showcasing the usage of `FL_+CALLBACK_*` in the examples/callbacks.cxx file.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIDGET</td>
<td>the widget that will call the callback</td>
</tr>
<tr>
<td>CLASS</td>
<td>the class type</td>
</tr>
<tr>
<td>SELF</td>
<td>a pointer to an instance of the class</td>
</tr>
<tr>
<td>METH</td>
<td>a C++ class method that must be public and not static</td>
</tr>
<tr>
<td>TYPE0,VALUE0</td>
<td>a list of zero to five type/value pairs, all separated by commas</td>
</tr>
</tbody>
</table>

### See also

`FL_FUNCTION_CALLBACK_3`, `FL_INLINE_CALLBACK_2`

### 13.25 fl_callback_macros.H

Go to the documentation of this file.
These macros create boilerplate code for callbacks to functions and static class methods with up to five arguments.

This macro invocation for example

```
FL_FUNCTION_CALLBACK_2(func_cb_btn_2, hello_2_args_cb,
    const char *, text, "FLTK",
    int, number, 2);
```

will generate the following code:

```
do {
    class Fl_Callback_User_Data_92 : public Fl_Callback_User_Data {
        public:
            const char * p0_; int p1_;
        static void cb(Fl_Widget *w, void *user_data) {
            Fl_Callback_User_Data_92 *d = (Fl_Callback_User_Data_92*)user_data;
            hello_2_args_cb(d->p0_, d->p1_);
        }
        Fl_Callback_User_Data_92(const char * p0, int p1)
            : p0_(p0), p1_(p1) {
    };
    func_cb_btn_2->callback(Fl_Callback_User_Data_92::cb,
        new Fl_Callback_User_Data_92("FLTK", 2),
        true);
} while(0)
```

Clicking the Fl_Button `func_cb_btn_2' will call `hello_2_args_cb("FLTK", 2)'.

Deleting the button will also delete the data that was created in our boilerplate code.

```
#define FL_FUNCTION_CALLBACK_5(WIDGET, FUNC, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2, TYPE3, VALUE3, TYPE4, VALUE4)
    do {
        class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data {
            public:
                TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; TYPE4 p4_; 
        static void cb(Fl_Widget *w, void *user_data) {
            _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data;
            FUNC(d->p0_, d->p1_, d->p2_, d->p3_, d->p4_);
        }
        _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1, TYPE2 p2, TYPE3 p3, TYPE4 p4)
            : p0_(p0), p1_(p1), p2_(p2), p3_(p3), p4_(p4) {};
        };
        WIDGET->callback(_FL_CBD_CLASS_NAME::cb,
            new _FL_CBD_CLASS_NAME(VALUE0, VALUE1, VALUE2, VALUE3, VALUE4), true);
    } while(0)
```

```
#define FL_FUNCTION_CALLBACK_4(WIDGET, FUNC, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2, TYPE3, VALUE3)
    do {
        class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data {
            public:
                TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; 
        static void cb(Fl_Widget *w, void *user_data) {
            _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data;
            FUNC(d->p0_, d->p1_, d->p2_, d->p3_);
        }
        _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1, TYPE2 p2, TYPE3 p3)
            : p0_(p0), p1_(p1), p2_(p2), p3_(p3) {};
        };
        WIDGET->callback(_FL_CBD_CLASS_NAME::cb,
            new _FL_CBD_CLASS_NAME(VALUE0, VALUE1, VALUE2, VALUE3), true);
    } while(0)
```

```
#define FL_FUNCTION_CALLBACK_3(WIDGET, FUNC, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2)
    do {
        class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data {
            public:
                TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; 
        static void cb(Fl_Widget *w, void *user_data) {
            _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data;
            FUNC(d->p0_, d->p1_, d->p2_);
        }
        _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1, TYPE2 p2)
            : p0_(p0), p1_(p1), p2_(p2) {};
    };
```

Generated by Doxygen
00272 WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1, VALUE2), true); \
00273 } while(0);  
00274  
00275 #define FL_FUNCTION_CALLBACK_2(WIDGET, FUNC, TYPES, VALUE0, TYPE1, VALUE1) \  
00276 do { \  
00277 class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \  
00278 public: \  
00279 TYPES p0_, TYPE1 p1_; \  
00280 static void cb(Fl_Widget *w, void *user_data) { \  
00281 _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \  
00282 FUNC(d->p0_, d->p1_); \  
00283 }; \  
00284 _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1) \  
00285 : p0_(p0), p1_(p1) { }; \  
00286 \  
00287 WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1), true); \  
00288 } while(0); \  
00289  
00290 #define FL_FUNCTION_CALLBACK_1(WIDGET, FUNC, TYPE0, VALUE0) \  
00291 do { \  
00292 class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \  
00293 public: \  
00294 TYPE0 p0_; \  
00295 static void cb(Fl_Widget *w, void *user_data) { \  
00296 _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \  
00297 FUNC(d->p0_); \  
00298 }; \  
00299 _FL_CBD_CLASS_NAME(TYPE0 p0) \  
00300 : p0_(p0) { }; \  
00301 \  
00302 WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0), true); \  
00303 } while(0); \  
00304  
00305 #define FL_FUNCTION_CALLBACK_0(WIDGET, FUNC) \  
00306 do { \  
00307 class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \  
00308 public: \  
00309 static void cb(Fl_Widget *w, void *user_data) { \  
00310 FUNC(); \  
00311 }; \  
00312 _FL_CBD_CLASS_NAME(); { }; \  
00313 \  
00314 WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(), true); \  
00315 } while(0); \  
00316 \  
00317 /*\  
00318 These macros create boilerplate code for callbacks to class methods\ 00319 with up to five arguments.\ 00320 \ 00321 This macro invocation for example\ 00322 \ 00323 FL_METHOD_CALLBACK_4(btn,\ 00324 MyWindow, win, resize,\ 00325 int, test_x+10,\ 00326 int, test_y+10,\ 00327 int, 320,\ 00328 int, 400);\ 00329 \ 00330 will generate the following code:\ 00331 ...\ 00332 \ 00333 do { \ 00334 class Fl_Callback_User_Data_73 : public Fl_Callback_User_Data { \ 00335 public: \ 00336 int p0_; \ 00337 int p1_; \ 00338 int p2_; \ 00339 int p3_; \ 00340 MyWindow *self_; \ 00341 static void cb(Fl_Widget *w, void *user_data) { \ 00342 Fl_Callback_User_Data_73 *d = (Fl_Callback_User_Data_73*)user_data; \ 00343 d->self_->_resize(d->p0_, d->p1_, d->p2_, d->p3_); \ 00344 }; \ 00345 Fl_Callback_User_Data_73(MyWindow *self, int p0, int p1, int p2, int p3) \ 00346 : self_(self), p0_(p0), p1_(p1), p2_(p2), p3_(p3) { } \ 00347 }; \ 00348 btn->callback(Fl_Callback_User_Data_73::cb, \ 00349 new Fl_Callback_User_Data_73(win, test_x+10, test_y+10, 320, 400), \ 00350 true); \ 00351 } while(0); \ 00352 ...\ 00353 \ 00354 Clicking the Fl_Button 'btn' will call\ 00355 'win->resize(test_x+10, test_y+10, 320, 400);'.\ 00356 Deleting the button will also delete the data that was created in our\ 00357 boilerplate code.
00358 */
#define FL_METHOD_CALLBACK_5(WIDGET, CLASS, SELF, METHOD, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2, TYPE3, VALUE3, TYPE4, VALUE4) 
{
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { 
    public: 
        CLASS *self_; 
        TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; TYPE4 p4_; 
    static void cb(Fl_Widget *w, void *user_data) { 
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; 
        d->self_->METHOD(d->p0_, d->p1_, d->p2_, d->p3_, d->p4_); 
    } 
    
    _FL_CBD_CLASS_NAME(CLASS *self, TYPE0 p0, TYPE1 p1, TYPE2 p2, TYPE3 p3, TYPE4 p4) 
    : self_(self), p0_(p0), p1_(p1), p2_(p2), p3_(p3), p4_(p4) { } 
    
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(SELF, VALUE0, VALUE1, VALUE2, VALUE3, VALUE4), true); 
} 

#define FL_METHOD_CALLBACK_4(WIDGET, CLASS, SELF, METHOD, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2, TYPE3, VALUE3) 
{
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { 
    public: 
        CLASS *self_; 
        TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; 
    static void cb(Fl_Widget *w, void *user_data) { 
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; 
        d->self_->METHOD(d->p0_, d->p1_, d->p2_, d->p3_); 
    } 
    
    _FL_CBD_CLASS_NAME(CLASS *self, TYPE0 p0, TYPE1 p1, TYPE2 p2, TYPE3 p3) 
    : self_(self), p0_(p0), p1_(p1), p2_(p2), p3_(p3) { } 
    
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(SELF, VALUE0, VALUE1, VALUE2, VALUE3), true); 
} 

#define FL_METHOD_CALLBACK_3(WIDGET, CLASS, SELF, METHOD, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2) 
{
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { 
    public: 
        CLASS *self_; 
        TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; 
    static void cb(Fl_Widget *w, void *user_data) { 
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; 
        d->self_->METHOD(d->p0_, d->p1_, d->p2_); 
    } 
    
    _FL_CBD_CLASS_NAME(CLASS *self, TYPE0 p0, TYPE1 p1, TYPE2 p2) 
    : self_(self), p0_(p0), p1_(p1), p2_(p2) { } 
    
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(SELF, VALUE0, VALUE1, VALUE2), true); 
} 

#define FL_METHOD_CALLBACK_2(WIDGET, CLASS, SELF, METHOD, TYPE0, VALUE0, TYPE1, VALUE1) 
{
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { 
    public: 
        CLASS *self_; 
        TYPE0 p0_; TYPE1 p1_; 
    static void cb(Fl_Widget *w, void *user_data) { 
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; 
        d->self_->METHOD(d->p0_, d->p1_); 
    } 
    
    _FL_CBD_CLASS_NAME(CLASS *self, TYPE0 p0, TYPE1 p1) 
    : self_(self), p0_(p0), p1_(p1) { } 
    
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(SELF, VALUE0, VALUE1), true); 
} 

#define FL_METHOD_CALLBACK_1(WIDGET, CLASS, SELF, METHOD, TYPE0, VALUE0) 
{
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { 
    public: 
        CLASS *self_; 
        TYPE0 p0_; 
    static void cb(Fl_Widget *w, void *user_data) { 
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; 
        d->self_->METHOD(d->p0_); 
    } 
    
    _FL_CBD_CLASS_NAME(CLASS *self, TYPE0 p0) 
    : self_(self), p0_(p0) { } 
    
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(SELF, VALUE0), true); 
}
```c
#define FL_METHOD_CALLBACK_0(WIDGET, CLASS, SELF, METHOD) \
  do { \n    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \n      public: \n        CLASS *self_; \n        static void cb(Fl_Widget *w, void *user_data) { \n          _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \n          d->self_->METHOD(); \n        } \n        _FL_CBD_CLASS_NAME(CLASS *self) : self_(self) { } \n    }; \n    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(SELF), true); \n  } while(0) \n/*
These macros create boilerplate code for callback functions inlined into
the widget creation code (similar to lambda functions in C++11 and up)
with up to five arguments.

This macro invocation for example

```
#define FL_INLINE_CALLBACK_2( // callback has two parameters
  WIDGET, TYPE0, NAME0, VALUE0, TYPE1, NAME1, VALUE1, LAMBDA) \
  do { \n    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \n      public: \n        TYPE0 p0_; TYPE1 p1_; static void fn(TYPE0 NAME0, TYPE1 NAME1) LAMBDA; \n        static void cb(Fl_Widget *w, void *user_data) { \n          _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \n          d->fn(d->p0_, d->p1_); \n        } \n        _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1) : p0_(p0), p1_(p1) { } \n    }; \n    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1), true); \n  } while(0); \n```

Clicking the Fl_Button `btn' will call
`fl_message("We received the message %s with %d!", "FLTK", 2);'.
Deleting the button will also delete the data that was created in our
boilerplate code.

/ *
#define FL_INLINE_CALLBACK_4(WIDGET, TYPE0, NAME0, VALUE0, TYPE1, NAME1, VALUE1, TYPE2, NAME2, VALUE2,
TYPE3, NAME3, VALUE3, LAMBDA) \
  do { \n    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \n      public: \n        TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; TYPE4 p4_; static void fn(TYPE0 NAME0, TYPE1 NAME1, TYPE2 NAME2, TYPE3 NAME3, TYPE4 NAME4) LAMBDA; \n        static void cb(Fl_Widget *w, void *user_data) { \n          _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \n          d->fn(d->p0_, d->p1_, d->p2_, d->p3_, d->p4_); \n        } \n        _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1, TYPE2 p2, TYPE3 p3, TYPE4 p4) : p0_(p0), p1_(p1), p2_(p2), p3_(p3), p4_(p4) { } \n    }; \n    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1, VALUE2, VALUE3, VALUE4), true); \n  } while(0) \n#define FL_INLINE_CALLBACK_5(WIDGET, TYPE0, NAME0, VALUE0, TYPE1, NAME1, VALUE1, TYPE2, NAME2, VALUE2,
TYPE3, NAME3, VALUE3, TYPE4, NAME4, VALUE4, LAMBDA) \
  do { \n    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \n      public: \n        TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; TYPE4 p4_; TYPE5 p5_; TYPE6 p6_; TYPE7 p7_; TYPE8 p8_;
        static void fn(TYPE0 NAME0, TYPE1 NAME1, TYPE2 NAME2, TYPE3 NAME3, TYPE4 NAME4, TYPE5 NAME5, TYPE6 NAME6, TYPE7 NAME7, TYPE8 NAME8) LAMBDA; \n        static void cb(Fl_Widget *w, void *user_data) { \n          _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \n          d->fn(d->p0_, d->p1_, d->p2_, d->p3_, d->p4_, d->p5_, d->p6_, d->p7_, d->p8_); \n        } \n        _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1, TYPE2 p2, TYPE3 p3, TYPE4 p4, TYPE5 p5, TYPE6 p6, TYPE7 p7, TYPE8 p8) : p0_(p0), p1_(p1), p2_(p2), p3_(p3), p4_(p4), p5_(p5), p6_(p6), p7_(p7), p8_(p8) { } \n    }; \n    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1, VALUE2, VALUE3, VALUE4, VALUE5, VALUE6, VALUE7, VALUE8), true); \n  } while(0)
*/
```
do {

class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data {

public:

    TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; 

static void fn(TYPE0 NAME0, TYPE1 NAME1, TYPE2 NAME2, TYPE3 NAME3) {
    LAMBDA; 
}

static void cb(Fl_Widget *w, void *user_data) {

    _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; 

    _FL_CBD_CLASS_NAME::fn(d->p0_, d->p1_, d->p2_, d->p3_); 
}

};

_WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1, VALUE2, VALUE3), true); 

} while(0)

#define FL_INLINE_CALLBACK_3(WIDGET, TYPE0, NAME0, VALUE0, TYPE1, NAME1, VALUE1, TYPE2, NAME2, VALUE2, LAMBDA) 

do {

class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data {

public:

    TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; 

static void fn(TYPE0 NAME0, TYPE1 NAME1, TYPE2 NAME2) {
    LAMBDA; 
}

static void cb(Fl_Widget *w, void *user_data) {

    _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; 

    _FL_CBD_CLASS_NAME::fn(d->p0_, d->p1_, d->p2_, d->p3_); 
}

};

_WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1, VALUE2, VALUE3), true); 

} while(0)

#define FL_INLINE_CALLBACK_2(WIDGET, TYPE0, NAME0, VALUE0, TYPE1, NAME1, VALUE1, LAMBDA) 

do {

class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data {

public:

    TYPE0 p0_; TYPE1 p1_; 

static void fn(TYPE0 NAME0, TYPE1 NAME1) {
    LAMBDA; 
}

static void cb(Fl_Widget *w, void *user_data) {

    _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; 

    _FL_CBD_CLASS_NAME::fn(d->p0_, d->p1_); 
}

};

_WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1), true); 

} while(0)

#define FL_INLINE_CALLBACK_1(WIDGET, TYPE0, NAME0, VALUE0, LAMBDA) 

do {

class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data {

public:

    TYPE0 p0_; 

static void fn(TYPE0 NAME0) {
    LAMBDA; 
}

static void cb(Fl_Widget *w, void *user_data) {

    _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; 

    _FL_CBD_CLASS_NAME::fn(d->p0_); 
}

};

_WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0), true); 

} while(0)

#define FL_INLINE_CALLBACK_0(WIDGET, LAMBDA) 

do {

class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data {

public:

static void fn() {
    LAMBDA; 
}

static void cb(Fl_Widget *w, void *user_data) {

    _FL_CBD_CLASS_NAME::fn(); 
}

};

_WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(), true); 

} while(0)

#endif // FL_DOXYGEN

#endif /* !_FL_FL_CALLBACK_MACROS_H_ */

Generated by Doxygen
13.26 fl_casts.H

```c
#ifndef _FL_fl_casts_H_
#define _FL_fl_casts_H_

#include <FL/platform_types.h>

inline char fl_char(void *v) { return (char)(fl_intptr_t)v; }
inline int fl_int(void *v) { return (int)(fl_intptr_t)v; }
inline long fl_long(void *v) { return (long)(fl_intptr_t)v; }

inline unsigned char fl_uchar(void *v) { return (unsigned char)(fl_uintptr_t)v; }
inline unsigned int fl_uint(void *v) { return (unsigned int)(fl_uintptr_t)v; }
inline unsigned long fl_ulong(void *v) { return (unsigned long)(fl_uintptr_t)v; }

// the following conversions can be used to silence MSVC warning C4312:
// 'type cast': conversion from 'type' to 'void *' of greater size
inline void *fl_voidptr(int v) { return (void *)(fl_intptr_t)v; }
inline void *fl_voidptr(unsigned int v) { return (void *)(fl_uintptr_t)v; }
inline void *fl_voidptr(long v) { return (void *)(fl_intptr_t)v; }
inline void *fl_voidptr(unsigned long v) { return (void *)(fl_uintptr_t)v; }

#endif /* _FL_fl_casts_H_ */
```

13.27 Fl_Chart.H File Reference

**Fl_Chart** widget.

```
#include "Fl_Widget.H"
```

**Classes**

- class **Fl_Chart**
  
  `Fl_Chart` displays simple charts.

- struct **FL_CHART_ENTRY**
  
  For internal use only.

**Macros**

- `#define FL_BAR_CHART 0`
  
  `type()` for Bar Chart variant

- `#define FL_CHART_LABEL_MAX 18`
  
  max label length for entry

- `#define FL_CHART_MAX 128`
  
  max entries per chart

- `#define FL_FILL_CHART 3`
  
  `type()` for Fill Line Chart variant

- `#define FL_FILLED_CHART FL_FILL_CHART`
  
  for compatibility

- `#define FL_HORBAR_CHART 1`
  
  `type()` for Horizontal Bar Chart variant
• `#define FL_LINE_CHART` 2
type() for Line Chart variant

• `#define FL_PIE_CHART` 5
type() for Pie Chart variant

• `#define FL_SPECIALPIE_CHART` 6
type() for Special Pie Chart variant

• `#define FL_SPIKE_CHART` 4
type() for Spike Chart variant

### 13.27.1 Detailed Description

**Fl_Chart** widget.

### 13.28 Fl_Chart.H

Go to the documentation of this file.

```c
#ifndef Fl_Chart_H
#define Fl_Chart_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

// values for type()
#define FL_BAR_CHART 0
#define FL_HORBAR_CHART 1
#define FL_LINE_CHART 2
#define FL_FILL_CHART 3
#define FL_SPIKE_CHART 4
#define FL_PIE_CHART 5
#define FL_SPECIALPIE_CHART 6
#define FL_FILLED_CHART FL_FILL_CHART
#define FL_CHART_MAX 128
#define FL_CHART_LABEL_MAX 18

struct FL_CHART_ENTRY {
    float val;
    uchar col;
    char str[FL_CHART_LABEL_MAX + 1];
};

class FL_EXPORT Fl_Chart : public Fl_Widget {
    int numb;
    int maxnumb;
    int sizenumb;
    FL_CHART_ENTRY *entries;
    double min, max;
    uchar autosize_;
    Fl_Font textfont_;  // This font is used for drawing Text.
    Fl_Fontsize textsize_;  // This size is used for drawing Text.
    Fl_Color textcolor_;  // This color is used for drawing Text.
    protected:
    void draw() FL_OVERRIDE;

    // (static) protected draw methods (STR 2022)
    static void draw_barchart(int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[]);
};
```

Generated by Doxygen
double min, double max, int autosize, int maxnumb, Fl_Color textcolor);

static void draw_horbarchart(int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[],
  double min, double max, int autosize, int maxnumb, Fl_Color textcolor);

static void draw_linechart(int type, int x, int y, int w, int h, int numb,
  FL_CHART_ENTRY entries[], double min, double max, int autosize,
  int maxnumb, Fl_Color textcolor);

static void draw_piechart(int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[],
  int special, Fl_Color textcolor);

public:
  Fl_Chart(int X, int Y, int W, int H, const char *L = 0);
  ~Fl_Chart();
  void clear();
  void add(double val, const char *str = 0, unsigned col = 0);
  void insert(int ind, double val, const char *str = 0, unsigned col = 0);
  void replace(int ind, double val, const char *str = 0, unsigned col = 0);
  void bounds(double *a, double *b) const {
  *a = min;
  *b = max;
  }
  void bounds(double a, double b);
  int size() const { return numb; }
  int maxsize() const { return maxnumb; }
  Fl_Font textfont() const { return textfont_; }
  void textfont(Fl_Font s) { textfont_ = s; }
  Fl_Fontsize textsize() const { return textsize_; }
  void textsize(Fl_Fontsize s) { textsize_ = s; }
  Fl_Color textcolor() const { return textcolor_; }
  void textcolor(Fl_Color n) { textcolor_ = n; }
  uchar autosize() const { return autosize_; }
  void autosize(uchar n) { autosize_ = n; }

  void bounds(double a, double b);
#include "Fl_Browser_.H"

class FL_EXPORT Fl_Check_Browser : public Fl_Browser_ {

protected:

/* required routines for Fl_Browser_ subclass: */
void *item_first() const FL_OVERRIDE;
void *item_next(void *) const FL_OVERRIDE;
void *item_prev(void *) const FL_OVERRIDE;
int item_height(void *) const FL_OVERRIDE;
int item_width(void *) const FL_OVERRIDE;
void *item_draw(void *, int, int, int, int) const FL_OVERRIDE;
void item_select(void *, int) FL_OVERRIDE;
int item_selected(void *) const FL_OVERRIDE;
const char *item_text(void *) const FL_OVERRIDE;

public:

void *item_at(int index) const FL_OVERRIDE;
void item_swap(int ia, int ib);
void item_swap(void *, void *) FL_OVERRIDE;

/* private data */

#ifndef FL_DOXYGEN
struct cb_item {
    cb_item *next;
    cb_item *prev;
    char checked;
    char selected;
    char *text;
};
#endif // !FL_DOXYGEN

private:

cb_item *first;
cb_item *last;
cb_item *cache;
int nitems_;
int nchecked_;

private:

int find_item(int) const;
int lineno(cb_item *) const;

public:

Fl_Check_Browser(int x, int y, int w, int h, const char *l = 0);
~Fl_Check_Browser() { clear(); }
int add(char *s); // add an (unchecked) item
int add(char *s, int b); // add an item and set checked
// both return the new nitems()
int remove(int item); // delete an item. Returns nitems()
int add(const char *s) { return add((char *)s); }
int add(const char *s, int b) { return add((char *)s, b); }
void clear(); // delete all items
int nitems() const { return nitems_; }
int nchecked() const { return nchecked_; }
int checked(int item) const;
void checked(int item, int b);
void set_checked(int item);
void check_all();
void check_none();
int value() const; // currently selected item
char *text(int item) const; // returns pointer to internal buffer

protected:

int handle(int) FL_OVERRIDE;

#endif // Fl_Check_Browser_H

// IRIX 5.3 C++ compiler doesn’t support private structures...

#endif // !FL_DOXYGEN

private:

cb_item *first;
cb_item *last;
cb_item *cache;
int nitems_;
int nchecked_;

private:

int find_item(int) const;
int lineno(cb_item *) const;

public:

Fl_Check_Browser(int x, int y, int w, int h, const char *l = 0);
~Fl_Check_Browser() { clear(); }
int add(char *s); // add an (unchecked) item
int add(char *s, int b); // add an item and set checked
// both return the new nitems()
int remove(int item); // delete an item. Returns nitems()
int add(const char *s) { return add((char *)s); }
int add(const char *s, int b) { return add((char *)s, b); }
void clear(); // delete all items
int nitems() const { return nitems_; }
int nchecked() const { return nchecked_; }
int checked(int item) const;
void checked(int item, int b);
void set_checked(int item);
void check_all();
void check_none();
int value() const; // currently selected item
char *text(int item) const; // returns pointer to internal buffer

protected:

int handle(int) FL_OVERRIDE;

#endif // Fl_Check_Browser_H

13.30 Fl_Check_Button.H

// Check button header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2014 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in the file "COPYING" which should have been included with this file. If this file is missing or damaged, see the license at:

Generated by Doxygen

13.30 Fl_Check_Button.H
```cpp
#ifndef Fl_Check_Button_H
#define Fl_Check_Button_H

#include "Fl_Light_Button.H"

/*
class: Fl_Check_Button.
*/

class FL_EXPORT Fl_Check_Button : public Fl_Light_Button {
  public:
    Fl_Check_Button(int X, int Y, int W, int H, const char *L = 0);
};

#endif
```

```cpp
13.31 Fl_Choice.H

#ifndef Fl_Choice_H
#define Fl_Choice_H

#include "Fl_Menu_.H"

class FL_EXPORT Fl_Choice : public Fl_Menu_ {
  protected:
    void draw() FL_OVERRIDE;
  public:
    int handle(int) FL_OVERRIDE;

    Fl_Choice(int X, int Y, int W, int H, const char *L = 0);

    int value() const {return Fl_Menu_::value();}
    int value(int v);
    int value(const Fl_Menu_Item * v);
};

#endif
```

```cpp
13.32 Fl_Clock.H

#ifndef Fl_Clock_H
#define Fl_Clock_H

#include "Fl_Bounded.H"

class FL_EXPORT Fl_Clock {
  protected:
    void draw() FL_OVERRIDE;
  public:
    void start();
    void stop();
    Fl_Clock(int X, int Y, int W, int H, const char *L = 0);

    int value() const {return Fl_Bounded::value();}
    int value(int v);
    int value(const Fl_Bounded_Value * v);
};
```

Generated by Doxygen
```c
/*
 * Fl_Clock, Fl_Clock_Output widgets . */

#ifndef Fl_Clock_H
#define Fl_Clock_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

// Values for type():
// Please change doxygen documentation below (class Fl_Clock_Output)
// accordingly as well when changing the following type values:
#define FL_SQUARE_CLOCK 0
#define FL_ROUND_CLOCK 1
#define FL_ANALOG_CLOCK FL_SQUARE_CLOCK
#define FL_DIGITAL_CLOCK FL_SQUARE_CLOCK

// fabien: Please keep the horizontal formatting of both images in class desc,
// don't lose vertical space for nothing!

class FL_EXPORT Fl_Clock_Output : public Fl_Widget {
    int hour_, minute_, second_;  
    ulong value_;   
    int shadow_; // draw shadows of hands
    void drawhands(Fl_Color,Fl_Color); // part of draw

protected:
    void draw() FL_OVERRIDE;
    void draw(int X, int Y, int W, int H);
public:
    Fl_Clock_Output(int X, int Y, int W, int H, const char *L = 0);
    void value(ulong v); // set to this Unix time
    void value(int H, int m, int s);

    ulong value() const {return value_;}
    int hour() const {return hour_;
    int minute() const {return minute_;
    int second() const {return second_;

    void shadow(int mode) { shadow_ = mode ? 1 : 0; }

    Fl_Clock_Output(int X, int Y, int W, int H, const char *L = 0);  
    void draw(int X, int Y, int W, int H, const char *L = 0);  
    ~Fl_Clock();
};

// a Fl_Clock displays the current time always by using a timeout:
class FL_EXPORT Fl_Clock : public Fl_Clock_Output {
public:
    int handle(int) FL_OVERRIDE;
    Fl_Clock(int X, int Y, int W, int H, const char *L = 0);
    Fl_Clock(uchar t, int X, int Y, int W, int H);
    ~Fl_Clock();
};
#endif
```

13.33 Fl_Color_Chooser.H File Reference

Fl_Color_Chooser widget .

```c
#include <FL/Fl_Group.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Return_Button.H>
#include <FL/Fl_Choice.H>
#include <FL/Fl_Value_Input.H>
```
Classes

- class Fl_Color_Chooser

  The Fl_Color_Chooser widget provides a standard RGB color chooser.

13.33.1 Detailed Description

Fl_Color_Chooser widget.

13.34 Fl_Color_Chooser.H

Go to the documentation of this file.

```cpp
// Color chooser header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2019 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php

// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#ifndef Fl_Color_Chooser_H
#define Fl_Color_Chooser_H

#include <FL/Fl_Group.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Return_Button.H>
#include <FL/Fl_Choice.H>
#include <FL/Fl_Value_Input.H>

#ifndef FL_DOXYGEN

class FL_EXPORT Flcc_HueBox : public Fl_Widget {
    int px, py;

protected:
    void draw() FL_OVERRIDE;
    int handle_key(int);

public:
    int handle(int) FL_OVERRIDE;
    Flcc_HueBox(int X, int Y, int W, int H) : Fl_Widget(X,Y,W,H) {
        px = py = 0;
    }
};

class FL_EXPORT Flcc_ValueBox : public Fl_Widget {
    int py;

protected:
    void draw() FL_OVERRIDE;
    int handle_key(int);

public:
    int handle(int) FL_OVERRIDE;
    Flcc_ValueBox(int X, int Y, int W, int H) : Fl_Widget(X,Y,W,H) {
        py = 0;
    }
};

class FL_EXPORT Flcc_Value_Input : public Fl_Value_Input {
    int format(char *) FL_OVERRIDE;
    Flcc_Value_Input(int X, int Y, int W, int H) : Fl_Value_Input(X,Y,W,H) {}
};
#endif // !FL_DOXYGEN

class FL_EXPORT Fl_Color_Chooser : public Fl_Group {
    Flcc_HueBox huebox;
    Flcc_ValueBox valuebox;
    Fl_Choice choice;
    Flcc_Value_Input rvalue;
    Flcc_Value_Input gvalue;
    Flcc_Value_Input bvalue;
    Fl_Box resize_box;
```

Generated by Doxygen
double hue_, saturation_, value_;  
void set_valuators();  
static void rgb_cb(Fl_Widget*, void*);  
static void mode_cb(Fl_Widget*, void*);  
public:
  int handle(int e) FL_OVERRIDE;
  int mode() {return choice.value();}
  void mode(int newMode);
  double hue() const {return hue_}
  double saturation() const {return saturation_}
  double value() const {return value_}
  double r() const {return r_}
  double g() const {return g_}
  double b() const {return b_}
  double hsv(double H, double S, double V);
  int rgb(double R, double G, double B);
  static void hsv2rgb(double H, double S, double V, double& R, double& G, double& B);
  static void rgb2hsv(double R, double G, double B, double& H, double& S, double& V);
  Fl_Color_Chooser(int X, int Y, int W, int H, const char* = 0);
  FL_EXPORT int fl_color_chooser(const char*, double& r, double& g, double& b, int m=-1);
  FL_EXPORT int fl_color_chooser(const char*, uchar& r, uchar& g, uchar& b, int m=-1);
  #endif

/* Get FL/fl_config.h. Generated from fl_config.in by configure. */
/* Build configuration file for the Fast Light Tool Kit (FLTK).*/
/* Copyright 1998-2023 by Bill Spitzak and others.*/
/* This library is free software. Distribution and use rights are outlined in*/
/* the file "COPYING" which should have been included with this file. If this*/
/* file is missing or damaged, see the license at:*/
/* https://www.fltk.org/COPYING.php*/
/* See the following page on how to report bugs and issues:*/
/* https://www.fltk.org/bugs.php*/
/* */
/* #ifndef _FL_fl_config_h_*/
/* #define _FL_fl_config_h_*/
/* */
/* #ifndef _FL_ABI_VERSION */
/* #define _FL_ABI_VERSION */
/* */
/* #ifndef FL_ABI_VERSION */
/* #define FL_ABI_VERSION */
/* */
/* #ifdef FLTK_HAVE_CAIRO */
/* #define FLTK_HAVE_CAIRO */
/* */
/* #ifdef FLTK_HAVE_CAIROEXT */
/* #define FLTK_HAVE_CAIROEXT */
/* */
/* #ifdef FLTK_HAVE_CAIRO */
/* #define FLTK_HAVE_CAIRO */
/* */
/* Generated by Doxygen*/
This implies to link cairo.lib in all FLTK based apps.
/*
 * #undef FLTK_HAVE_CAIROEXT */

/*
 * FLTK_HAVE_FORMS
 *
 * Do we have the Forms compatibility library available?
 */
#define FLTK_HAVE_FORMS 1

/*
 * FLTK_USE_X11
 *
 * Do we use X11 for the current platform?
 */
#define FLTK_USE_X11 1

/*
 * FLTK_USE_CAIRO
 *
 * Do we use Cairo to draw to the display?
 */
#define FLTK_USE_CAIRO 1

/*
 * FLTK_USE_WAYLAND
 *
 * Do we use Wayland for the current platform?
 */
#define FLTK_USE_WAYLAND 1

/*
 * FLTK_USE_STD
 *
 * May we use std::string and std::vector for the current build?
 */
#define FLTK_USE_STD 0

#endif /* _FL_fl_config_h_ */

13.36 Fl_Copy_Surface.H

// Copy-to-clipboard code for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2023 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file COPYING which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#ifndef Fl_Copy_Surface_H
#define Fl_Copy_Surface_H

#include <FL/Fl_Widget_Surface.H>

class FL_EXPORT Fl_Copy_Surface : public Fl_Widget_Surface {
 private:
  class Fl_Copy_Surface_Driver *platform_surface;

  // This is a build configuration option which allows FLTK to add some
  // features based on std::string and std::vector in FLTK 1.4.x
  #define FLTK_USE_STD 0

  // This implies to link cairo.lib in all FLTK based apps.
  /*
   * #undef FLTK_HAVE_CAIROEXT */

  /*
   * FLTK_HAVE_FORMS
   *
   * Do we have the Forms compatibility library available?
   */
  #define FLTK_HAVE_FORMS 1

  /*
   * FLTK_USE_X11
   *
   * Do we use X11 for the current platform?
   */
  #define FLTK_USE_X11 1

  /*
   * FLTK_USE_CAIRO
   *
   * Do we use Cairo to draw to the display?
   */
  #define FLTK_USE_CAIRO 1

  /*
   * FLTK_USE_WAYLAND
   *
   * Do we use Wayland for the current platform?
   */
  #define FLTK_USE_WAYLAND 1

  /*
   * FLTK_USE_STD
   *
   * May we use std::string and std::vector for the current build?
   */
  #define FLTK_USE_STD 0

  #endif /* _FL_fl_config_h_ */
```cpp
protected:
void translate(int x, int y) FL_OVERRIDE;
void untranslate() FL_OVERRIDE;

public:
Fl_Copy_Surface(int w, int h);

Fl_Copy_Surface_Driver(int w, int h) : Fl_Widget_Surface(NULL), width(w), height(h) {}
virtual ~Fl_Copy_Surface_Driver() {};

void set_current() FL_OVERRIDE = 0;
void translate(int x, int y) FL_OVERRIDE = 0;
void untranslate() FL_OVERRIDE = 0;

int printable_rect(int *w, int *h) FL_OVERRIDE;
static Fl_Copy_Surface_Driver *newCopySurfaceDriver(int w, int h);

};

//
// Counter header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2022 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#define FL_NORMAL_COUNTER 0
#define FL_SIMPLE_COUNTER 1

class Fl_Counter : public Fl_Valuator {

};

Generated by Doxygen
```
Fl_Counter(int X, int Y, int W, int H, const char * L = 0);
~Fl_Counter();

void lstep(double a) {lstep_ = a;}

void step(double a, double b) {Fl_Valuator::step(a); lstep_ = b;}

void step(double a) {Fl_Valuator::step(a);}

double step() const {return Fl_Valuator::step();}

Fl_Font textfont() const {return textfont_;
void textfont(Fl_Font s) {textfont_ = s;

Fl_Fontsize textsize() const {return textsize_;
void textsize(Fl_Fontsize s) {textsize_ = s;

Fl_Color textcolor() const {return textcolor_;
void textcolor(Fl_Color s) {textcolor_ = s;

};

#endif

13.38 Fl_Device.H File Reference

declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin.
#include <FL/Fl_Plugin.H>
#include <FL/platform_types.h>

Classes

- class Fl_Device_Plugin
  This plugin socket allows the integration of new device drivers for special window or screen types.
- class Fl_Display_Device
  The computer's display.
- class Fl_Surface_Device
  A drawing surface that's susceptible to receive graphical output.

13.38.1 Detailed Description

declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin.

13.39 Fl_Device.H

Go to the documentation of this file.
class Fl_RGB_Image;
class Fl_Widget;
class Fl_Image_Surface;

class FL_EXPORT Fl_Surface_Device {
Fl_Graphics_Driver *pGraphicsDriver;
static Fl_Surface_Device *surface_; // the surface that currently receives graphics requests
static Fl_Surface_Device *default_surface(); // create surface if none exists yet
protected:
virtual void end_current() { surface_ = 0; }
inline void driver(Fl_Graphics_Driver *graphics_driver) { pGraphicsDriver = graphics_driver; }
inline Fl_Surface_Device *default_surface();
protected:
virtual void set_current(void);
virtual bool is_current();
inline Fl_Graphics_Driver *driver() { return pGraphicsDriver; }
public:
virtual ~Fl_Surface_Device();
static void push_current(Fl_Surface_Device *new_current);
static Fl_Surface_Device *pop_current();
};

class FL_EXPORT Fl_Display_Device : public Fl_Surface_Device {
Fl_Display_Device(Fl_Graphics_Driver *graphics_driver);
public:
static Fl_Display_Device *display_device();
};

class Fl_Device_Plugin : public Fl_Plugin {
 Fl_Device_Plugin(const char *pluginName)
: Fl_Plugin(klass(), pluginName) { }
virtual const char *klass() { return "fltk:device"; }
virtual const char *name() = 0;
virtual int print(Fl_Widget * w) = 0;
virtual Fl_RGB_Image * rectangle_capture(Fl_Widget *widget, int x, int y, int w, int h) = 0;
static Fl_Device_Plugin *opengl_plugin();
};

#endif // Fl_Device_H

Fl_Dial.H

Fl_Dial widget.

#ifndef Fl_Dial_H
#define Fl_Dial_H

#ifndef Fl_Valuator_H
#include "Fl_Valuator.H"
#endif

// values for type():
#define FL_NORMAL_DIAL 0
#define FL_LINE_DIAL 1
#define FL_FILL_DIAL 2

class FL_EXPORT Fl_Dial : public Fl_Valuator {
short a1,a2;
protected:
#endif // Fl_Dial_H
```cpp
int handle(int) FL_OVERRIDE;
Fl_Dial(int x, int y, int w, int h, const char *l = 0);
short angle1() const {return a1;}
void angle1(short a) {a1 = a;}
short angle2() const {return a2;}
void angle2(short a) {a2 = a;}
void angles(short a, short b) {a1 = a; a2 = b;}

Fl_Double_Window(int W, int H, const char *l = 0);
Fl_Double_Window(int X, int Y, int W, int H, const char *l = 0);
```

### 13.41 Fl_Double_Window.H

```cpp
Fl_Double_Window *as_double_window() FL_OVERRIDE {return this; }
void show() FL_OVERRIDE;
void show(int a, char **b) {Fl_Window::show(a,b);}
void resize(int,int,int,int) FL_OVERRIDE;
void hide() FL_OVERRIDE;
void flush() FL_OVERRIDE;
~Fl_Double_Window();
Fl_Double_Window(int W, int H, const char *l = 0);
Fl_Double_Window(int X, int Y, int W, int H, const char *l = 0);
```

### 13.42 fl_draw.H File Reference

Utility header to pull drawing functions together

```cpp
#include <FL/Enumerations.H>
#include <FL/Fl_Graphics_Driver.H>
#include <FL/Fl_Rect.H>
```

**Enumerations**

```cpp
enum {
    FL_SOLID = 0, FL_DASH = 1, FL_DOT = 2, FL_DASHDOT = 3,
    FL_DASHDOTDOT = 4, FL_CAP_FLAT = 0x100, FL_CAP_ROUND = 0x200, FL_CAP_SQUARE = 0x300,
    FL_JOIN_MITER = 0x1000, FL_JOIN_ROUND = 0x2000, FL_JOIN_BEVEL = 0x3000
}
```
Functions

- `int fl_add_symbol (const char *name, void(*)(Fl_Color), int scalable)`
  
  Adds a symbol to the system.

- `int fl_antialias ()`
  
  Return whether line drawings are currently antialiased.

- `void fl_antialias(int state)`
  
  Turn antialiased line drawings ON or OFF, if supported by platform.

- `void fl_arc (double x, double y, double r, double start, double end)`
  
  Add a series of points to the current path on the arc of a circle.

- `void fl_arc (int x, int y, int w, int h, double a1, double a2)`
  
  Draw ellipse sections using integer coordinates.

- `void fl_begin_complex_polygon ()`
  
  Start drawing a complex filled polygon.

- `void fl_begin_line ()`
  
  Start drawing a list of lines.

- `void fl_begin_loop ()`
  
  Start drawing a closed sequence of lines.

- `void fl_begin_offscreen (Fl_Offscreen ctx)`
  
  Send all subsequent drawing commands to this offscreen buffer.

- `void fl_begin_points ()`
  
  Start drawing a list of points.

- `void fl_begin_polygon ()`
  
  Start drawing a convex filled polygon.

- `char fl_can_do_alpha_blending ()`
  
  Check whether platform supports true alpha blending for RGBA images.

- `Fl_RGB_Image * fl_capture_window (Fl_Window *win, int x, int y, int w, int h)`
  
  Captures the content of a rectangular zone of a mapped window.

- `void fl_chord (int x, int y, int w, int h, double a1, double a2)`
  
  fl_chord declaration is a place holder - the function does not yet exist

- `void fl_circle (double x, double y, double r)`
  
  fl_circle(x,y,r) is equivalent to fl_arc(x,y,r,0,360), but may be faster.

- `void fl_clip (int x, int y, int w, int h)`
  
  Intersect the current clip region with a rectangle and push this new region onto the stack (deprecated).

- `int fl_clip_box (int x, int y, int w, int h, int &X, int &Y, int &W, int &H)`
  
  Intersect a rectangle with the current clip region and return the bounding box of the result.

- `Fl_Region fl_clip_region ()`
  
  Return the current clipping region.

- `void fl_clip_region (Fl_Region r)`
  
  Replace the top of the clipping stack with a clipping region of any shape.

- `Fl_Color fl_color ()`
  
  Return the last fl_color() that was set.

- `void fl_color (Fl_Color c)`
  
  Set the color for all subsequent drawing operations.

- `void fl_color (int c)`
  
  for back compatibility - use fl_color(Fl_Color c) instead

- `void fl_color (uchar r, uchar g, uchar b)`
  
  Set the color for all subsequent drawing operations.

- `void fl_copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)`
  
  Copy a rectangular area of the given offscreen buffer into the current drawing destination.

- `Fl_Offscreen fl_create_offscreen (int w, int h)`
Creation of an offscreen graphics buffer.

- `void fl_cursor (Fl_Cursor)`
  
  Sets the cursor for the current window to the specified shape and colors.

- `void fl_cursor (Fl_Cursor, Fl_Color fg, Fl_Color bg=FL_WHITE)`
  
  Add a series of points on a Bézier curve to the path.

- `void fl_delete_offscreen (Fl_Offscreen ctx)`
  
  Deletion of an offscreen graphics buffer.

- `int fl_descent ()`
  
  Return the recommended distance above the bottom of a fl_height() tall box to draw the text at so it looks centered vertically in that box.

- `void fl_draw (const char *str, int n, int x, int y)`
  
  Draws starting at the given x, y location a UTF-8 string of length n bytes.

- `void fl_draw (const char *str, int x, int y)`
  
  Draw a nul-terminated UTF-8 string starting at the given x, y location.

- `void fl_draw (const char *str, int x, int y, int w, int h, Fl_Align align, Fl_Image *img=0, int draw_symbols=1)`
  
  Fancy string drawing function which is used to draw all the labels.

- `void fl_draw (const char *str, int x, int y, int n, int w, int h, Fl_Align align, Fl_Image *img=0, int draw_symbols=1)`

  The same as fl_draw(const char *,int,int,int,Fl_Align,Fl_Image*,int) with the addition of the callthis parameter, which is a pointer to a text drawing function such as fl_draw(const char *, int, int, int) to do the real work.

- `void fl_draw (int angle, const char *str, int n, int x, int y)`
  
  Draw at the given x, y location a UTF-8 string of length n bytes rotating angle degrees counter-clockwise.

- `void fl_draw (int angle, const char *str, int x, int y)`
  
  Draw a nul-terminated UTF-8 string starting at the given x, y location and rotating angle degrees counter-clockwise.

- `void fl_draw_arrow (Fl_Rect bb, Fl_Arrow_Type t, Fl_Orientation o, Fl_Color color)`
  
  Draw an "arrow like" GUI element for the selected scheme.

- `void fl_draw_box (Fl_Boxtype, int x, int y, int w, int h, Fl_Color)`
  
  Draws a box using given type, position, size and color.

- `void fl_draw_check (Fl_Rect bb, Fl_Color col)`
  
  Draw a check mark inside the given bounding box.

- `void fl_draw_circle (int x, int y, int d, Fl_Color color)`
  
  Draw a potentially small, filled circle using a given color.

- `void fl_draw_image (const uchar *buf, int X, int Y, int W, int H, int D=3, int L=0)`
  
  Draw an 8-bit per color RGB or luminance image.

- `void fl_draw_image (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=3)`
  
  Draw an image using a callback function to generate image data.

- `void fl_draw_image_mono (const uchar *buf, int X, int Y, int W, int H, int D=1, int L=0)`
  
  Draw a gray-scale (1 channel) image.

- `void fl_draw_image_mono (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=1)`
  
  Draw a gray-scale (1 channel) image using a callback function to generate image data.

- `int fl_draw_pixmap (char const *data, int x, int y, Fl_Color bg=FL_GRAY)`
  
  Draw XPM image data, with the top-left corner at the given position.

- `int fl_draw_pixmap (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=1)`
  
  Draw XPM image data, with the top-left corner at the given position.

- `void fl_draw_radio (int x, int y, int d, Fl_Color color)`
  
  Draw a round check mark (circle) of a radio button.

- `int fl_draw_symbol (const char *label, int x, int y, int w, int h, Fl_Color)`

  Draw the named symbol in the given rectangle using the given color.

- `void fl_end_complex_polygon ()`

  End complex filled polygon, and draw.
- `void fl_end_line ()`
  End list of lines, and draw.
- `void fl_end_loop ()`
  End closed sequence of lines, and draw.
- `void fl_end_offscreen ()`
  Quit sending drawing commands to the current offscreen buffer.
- `void fl_end_points ()`
  End list of points, and draw.
- `void fl_end_polygon ()`
  End convex filled polygon, and draw.
- `const char * fl_expand_text (const char *from, char *buf, int maxbuf, double maxw, int &n, double &width, int wrap, int draw_symbols=0)`
  Copy from to buf, replacing control characters with ^X.
- `void fl_focus_rect (int x, int y, int w, int h)`
  Draw a dotted rectangle, used to indicate keyboard focus on a widget.
- `Fl_Font fl_font ()`
  Return the face set by the most recent call to fl_font().
- `void fl_font (Fl_Font face, Fl_Fontsize fsize)`
  Sets the current font, which is then used in various drawing routines.
- `void fl_frame (const char *s, int x, int y, int w, int h)`
  Draws a series of line segments around the given box.
- `void fl_frame2 (const char *s, int x, int y, int w, int h)`
  Draws a series of line segments around the given box.
- `void fl_gap ()`
  Separate loops of the path.
- `int fl_height ()`
  Return the recommended minimum line spacing for the current font.
- `int fl_height (int font, int size)`
  This function returns the actual height of the specified font and size.
- `const char * fl_latin1_to_local (const char *t, int n=-1)`
  Convert text from Windows/X11 latin1 character set to local encoding.
- `const char * fl_mac_roman_to_local (const char *t, int n=-1)`
  Convert text from Mac Roman character set to local encoding.
• void fl_measure (const char *str, int &x, int &y, int draw_symbols=1)
  Measure how wide and tall the string will be when printed by the fl_draw() function with align parameter.
• int fl_measure_pixmap (char *const *data, int &w, int &h)
  Get the dimensions of a pixmap.
• int fl_measure_pixmap (const char *const *cdata, int &w, int &h)
  Get the dimensions of a pixmap.
• void fl_mult_matrix (double a, double b, double c, double d, double x, double y)
  Concatenate another transformation onto the current one.
• int fl_not_clipped (int x, int y, int w, int h)
  Does the rectangle intersect the current clip region?
• unsigned int fl_oldShortcut (const char *s)
  Emulation of XForms named shortcuts.
• void fl_overlay_clear ()
  Erase a selection rectangle without drawing a new one.
• void fl_overlay_rect (int x, int y, int w, int h)
  Draw a transient dotted selection rectangle.
• float fl_override_scale ()
  Removes any GUI scaling factor in subsequent drawing operations.
• void fl_pie (int x, int y, int w, int h, double a1, double a2)
  Draw filled ellipse sections using integer coordinates.
• void fl_point (int x, int y)
  Draw a single pixel at the given coordinates.
• void fl_polygon (int x, int y, int x1, int y1, int x2, int y2)
  Fill a 3-sided polygon.
• void fl_polygon (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)
  Fill a 4-sided polygon.
• void fl_pop_clip ()
  Restore the previous clip region.
• void fl_pop_matrix ()
  Restore the current transformation matrix from the stack.
• void fl_push_clip (int x, int y, int w, int h)
  Intersect the current clip region with a rectangle and push this new region onto the stack.
• void fl_push_matrix ()
  Save the current transformation matrix on the stack.
• void fl_push_no_clip ()
  Push an empty clip region onto the stack so nothing will be clipped.
• uchar * fl_read_image (uchar *p, int X, int Y, int W, int H, int alpha=0)
  Reads an RGB(A) image from the current window or off-screen buffer.
• void fl_rect (Fl_Rect r)
  Draw a border inside the given bounding box.
• void fl_rect (int x, int y, int w, int h)
  Draw a border inside the given bounding box.
• void fl_rect (int x, int y, int w, int h, Fl_Color c)
  Draw with passed color a border inside the given bounding box.
• void fl_rectf (Fl_Rect bb, uchar r, uchar g, uchar b)
  Color a rectangle with "exactly" the passed r, g, b color.
• void fl_rectf (Fl_Rect r)
  Color with current color a rectangle that exactly fills the given bounding box.
• void fl_rectf (Fl_Rect r, Fl_Color c)
  Color with passed color a rectangle that exactly fills the given bounding box.
• void fl_rectf (int x, int y, int w, int h)
Color with current color a rectangle that exactly fills the given bounding box.

- `void fl_rectf (int x, int y, int w, int h, Fl_Color c)`

  Color with passed color a rectangle that exactly fills the given bounding box.

- `void fl_rectf (int x, int y, int w, int h, uchar r, uchar g, uchar b)`

  Color a rectangle with "exactly" the passed r, g, b color.

- `void fl_rescale_offscreen (Fl_Offscreen &ctx)`

  Adapts an offscreen buffer to a changed value of the scale factor.

- `void fl_reset_spot (void)`

  Resets marked text.

- `void fl_restore_clip ()`

  Undo any clobbering of the clip region done by your program.

- `void fl_restore_scale (float s)`

  Restores the GUI scaling factor and the clipping region in subsequent drawing operations.

- `void fl_rotate (double d)`

  Concatenate rotation transformation onto the current one.

- `void fl_rounded_rect (int x, int y, int w, int h, int r)`

  Draw a rounded border inside the given bounding box.

- `void fl_rounded_rectf (int x, int y, int w, int h, int r)`

  Color with current color a rounded rectangle that exactly fills the given bounding box.

- `void fl_rtl_draw (const char *str, int n, int x, int y)`

  Draw a UTF-8 string of length n bytes right to left starting at the given x, y location.

- `void fl_scale (double x)`

  Concatenate scaling transformation onto the current one.

- `void fl_scale (double x, double y)`

  Concatenate scaling transformation onto the current one.

- `void fl_scroll (int X, int Y, int W, int H, int dx, int dy, void (draw_area)(void *, int, int, int, int), void *data)`

  Scroll a rectangle and draw the newly exposed portions.

- `void fl_set_spot (int font, int size, int X, int Y, int W, int H, Fl_Window *win=0)`

  Inform text input methods about the current text insertion cursor.

- `void fl_set_status (int X, int Y, int W, int H)`

  Related to text input methods under X11.

- `const char *flShortcut_label (unsigned int shortcut)`

  Get a human-readable string from a shortcut value.

- `const char *flShortcut_label (unsigned int shortcut, const char **eom)`

  Get a human-readable string from a shortcut value.

- `Fl_Fontsize fl_size ()`

  Return the size set by the most recent call to fl_font().

- `void fl_text_extents (const char *, int &dx, int &dy, int &w, int &h)`

  Determine the minimum pixel dimensions of a null-terminated string using the current fl_font().

- `void fl_text_extents (const char *t, int n, int &dx, int &dy, int &w, int &h)`

  Determine the minimum pixel dimensions of a sequence of n characters (bytes) using the current fl_font().

- `double fl_transform_dx (double x, double y)`

  Transform distance using current transformation matrix.

- `double fl_transform_dy (double x, double y)`

  Transform distance using current transformation matrix.

- `double fl_transform_x (double x, double y)`

  Transform coordinate using the current transformation matrix.

- `double fl_transform_y (double x, double y)`

  Transform coordinate using the current transformation matrix.

- `void fl_transformed_vertex (double xf, double yf)`

  Add coordinate pair to the vertex list without further transformations.
• void fl_translate (double x, double y)
  Concatenate translation transformation onto the current one.

• void fl_vertex (double x, double y)
  Add a single vertex to the current path.

• double fl_width (const char ∗txt)
  Return the typographical width of a nul-terminated string using the current font face and size.

• double fl_width (const char ∗txt, int n)
  Return the typographical width of a sequence of n characters using the current font face and size.

• double fl_width (unsigned int c)
  Return the typographical width of a single character using the current font face and size.

• void fl_xyline (int x, int y, int x1)
  Draw a horizontal line from (x,y) to (x1,y).

• void fl_xyline (int x, int y, int x1, int y2)
  Draw a horizontal line from (x,y) to (x1,y), then vertical from (x1,y) to (x1,y2).

• void fl_xyline (int x, int y, int x1, int y2, int x3)
  Draw a horizontal line from (x,y) to (x1,y), then a vertical from (x1,y) to (x1,y2) and then another horizontal from (x1,y2) to (x3,y2).

• void fl_yxline (int x, int y, int y1)
  Draw a vertical line from (x,y) to (x,y1)

• void fl_yxline (int x, int y, int y1, int x2)
  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1).

• void fl_yxline (int x, int y, int y1, int x2, int y3)
  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1), then another vertical from (x2,y1) to (x2,y3).

Variables

• char fl_draw_shortcut

13.42.1 Detailed Description
utility header to pull drawing functions together

13.43 fl_draw.H

Go to the documentation of this file.
FL_EXPORT extern char fl_draw_shortcut;

// Colors:
inline void fl_color(Fl_Color c) { 
    fl_graphics_driver->color(c);
}
inline void fl_color(int c) { 
    fl_color((Fl_Color)c);
}
inline void fl_color(uchar r, uchar g, uchar b) { 
    fl_graphics_driver->color(r, g, b);
}
inline FL_Color fl_color() { 
    return fl_graphics_driver->color();
}

// clip:
inline void fl_push_clip(int x, int y, int w, int h) { 
    fl_graphics_driver->push_clip(x, y, w, h);
}
inline void fl_clip(int x, int y, int w, int h) { 
    fl_graphics_driver->push_clip(x, y, w, h);
}
inline void fl_push_no_clip() { 
    fl_graphics_driver->push_no_clip();
}
inline void fl_pop_clip() { 
    fl_graphics_driver->pop_clip();
}

inline int fl_not_clipped(int x, int y, int w, int h) { 
    return fl_graphics_driver->not_clipped(x, y, w, h);
}
inline int fl_clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H) { 
    return fl_graphics_driver->clip_box(x, y, w, h, X, Y, W, H);
}
inline void fl_restore_clip() { 
    fl_graphics_driver->restore_clip();
}
inline void fl_clip_region(Fl_Region r) { 
    fl_graphics_driver->clip_region(r);
}
inline Fl_Region fl_clip_region() { 
    return fl_graphics_driver->clip_region();
}

// points:
inline void fl_point(int x, int y) { 
    fl_graphics_driver->point(x, y);
}

// line type:
enum {
    FL_SOLID = 0,
    FL_DASH = 1,
    FL_DOT = 2,
    FL_DASHDOT = 3,
    FL_DASHDOTDOT = 4,
    FL_CAP_FLAT = 0x100,
    FL_CAP_ROUND = 0x200,
    FL_CAP_SQUARE = 0x300,
    FL_JOIN_MITER = 0x1000,
    FL_JOIN_ROUND = 0x2000,
    FL_JOIN_BEVEL = 0x3000
};

// rectangles tweaked to exactly fill the pixel rectangle:
inline void fl_rect(int x, int y, int w, int h) { 
    fl_graphics_driver->rect(x, y, w, h);
}
inline void fl_rounded_rect(int x, int y, int w, int h, int r) {
    fl_graphics_driver->rounded_rect(x, y, w, h, r);
}

inline void fl_rect(Fl_Rect r) {
    fl_rect(r.x(), r.y(), r.w(), r.h());
}

inline void fl_focus_rect(int x, int y, int w, int h) {
    fl_graphics_driver->focus_rect(x, y, w, h);
}

inline void fl_rect(int x, int y, int w, int h, Fl_Color c) {
    fl_color(c);
    fl_rect(x, y, w, h);
}

inline void fl_rectf(int x, int y, int w, int h) {
    fl_graphics_driver->rectf(x, y, w, h);
}

inline void fl_rounded_rectf(int x, int y, int w, int h, int r) {
    fl_graphics_driver->rounded_rectf(x, y, w, h, r);
}

inline void fl_rectf(int x, int y, int w, int h, Fl_Color c) {
    fl_color(c);
    fl_rectf(x, y, w, h);
}

inline void fl_rectf(Fl_Rect r) {
    fl_graphics_driver->rectf(r.x(), r.y(), r.w(), r.h());
}

inline void fl_rectf(Fl_Rect r, Fl_Color c) {
    fl_color(c);
    fl_rectf(r);
}

inline void fl_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b) {
    fl_graphics_driver->colored_rectf(x, y, w, h, r, g, b);
}

inline void fl_rectf(Fl_Rect bb, uchar r, uchar g, uchar b) {
    fl_graphics_driver->colored_rectf(bb.x(), bb.y(), bb.w(), bb.h(), r, g, b);
}

// line segments:
inline void fl_line(int x, int y, int x1, int y1) {
    fl_graphics_driver->line(x, y, x1, y1);
}

inline void fl_line(int x, int y, int x1, int y1, int x2, int y2) {
    fl_graphics_driver->line(x, y, x1, y1, x2, y2);
}

// closed line segments:
inline void fl_loop(int x, int y, int x1, int y1, int x2, int y2) {
    fl_graphics_driver->loop(x, y, x1, y1, x2, y2);
}

inline void fl_loop(int x, int y, int x1, int y1, int x2, int y2, int x3, int y3) {
    fl_graphics_driver->loop(x, y, x1, y1, x2, y2, x3, y3);
}

// filled polygons
inline void fl_polygon(int x, int y, int x1, int y1, int x2, int y2) {
    fl_graphics_driver->polygon(x, y, x1, y1, x2, y2);
}

inline void fl_polygon(int x, int y, int x1, int y1, int x2, int y2, int x3, int y3) {
    fl_graphics_driver->polygon(x, y, x1, y1, x2, y2, x3, y3);
}

// draw rectilinear lines, horizontal segment first:
inline void fl_xyline(int x, int y, int x1) {
    fl_graphics_driver->xyline(x, y, x1);
}

inline void fl_xyline(int x, int y, int x1, int y2) {
    fl_graphics_driver->xyline(x, y, x1, y2);
}

inline void fl_xyline(int x, int y, int x1, int y2, int x3) {
    fl_graphics_driver->xyline(x, y, x1, y2, x3);
}

// draw rectilinear lines, vertical segment first:
inline void fl_yxline(int x, int y, int y1) {
    fl_graphics_driver->yxline(x, y, y1);
}

inline void fl_yxline(int x, int y, int y1, int x2) {
    fl_graphics_driver->yxline(x, y, y1, x2);
}

inline void fl_yxline(int x, int y, int y1, int x2, int x3) {
    fl_graphics_driver->yxline(x, y, y1, x2, x3);
}
```c
inline void fl_yxline(int x, int y, int y1, int x2) {
    fl_graphics_driver->yxline(x, y, y1, x2);
}

inline void fl_yxline(int x, int y, int y1, int x2, int y3) {
    fl_graphics_driver->yxline(x, y, y1, x2, y3);
}

inline void fl_yxline(int x, int y, int y1, int x2) {
    fl_graphics_driver->yxline(x, y, y1, x2);
}
// circular lines and pie slices (code in fl_arci.C):
inline void fl_arc(int x, int y, int w, int h, double a1, double a2) {
    fl_graphics_driver->arc(x, y, w, h, a1, a2);
}
inline void fl_pie(int x, int y, int w, int h, double a1, double a2) {
    fl_graphics_driver->pie(x, y, w, h, a1, a2);
}
FL_EXPORT void fl_chord(int x, int y, int w, int h, double a1, double a2); // nyi
// scalable drawing code (code in fl_vertex.cxx and fl_arc.cxx):
inline void fl_push_matrix() {
    fl_graphics_driver->push_matrix();
}
inline void fl_pop_matrix() {
    fl_graphics_driver->pop_matrix();
}
inline void fl_scale(double x, double y) {
    fl_graphics_driver->mult_matrix(x, 0, 0, y, 0, 0);
}
inline void fl_scale(double x) {
    fl_graphics_driver->mult_matrix(x, 0, 0, x, 0, 0);
}
inline void fl_translate(double x, double y) {
    fl_graphics_driver->translate(x, y);
}
inline void fl_rotate(double d) {
    fl_graphics_driver->rotate(d);
}
inline void fl_load_identity() {
    fl_graphics_driver->load_identity();
}
inline void fl_load_matrix(double a, double b, double c, double d, double x, double y) {
    fl_graphics_driver->load_matrix(a, b, c, d, x, y);
}
inline void fl_mult_matrix(double a, double b, double c, double d, double x, double y) {
    fl_graphics_driver->mult_matrix(a, b, c, d, x, y);
}
inline void fl_begin_points() {
    fl_graphics_driver->begin_points();
}
inline void fl_begin_line() {
    fl_graphics_driver->begin_line();
}
inline void fl_begin_loop() {
    fl_graphics_driver->begin_loop();
}
inline void fl_begin_polygon() {
    fl_graphics_driver->begin_polygon();
}
inline void fl_vertex(double x, double y) {
    fl_graphics_driver->vertex(x, y);
}
inline void fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3,
        double Y3) {
    fl_graphics_driver->curve(X0, Y0, X1, Y1, X2, Y2, X3, Y3);
}
inline void fl_arc(double x, double y, double r, double start, double end) {
    fl_graphics_driver->arc(x, y, r, start, end);
}
inline void fl_circle(double x, double y, double r) {
    fl_graphics_driver->circle(x, y, r);
}
inline void fl_end_points() {
    fl_graphics_driver->end_points();
}
inline void fl_end_line() {
    fl_graphics_driver->end_line();
}
inline void fl_end_loop() {
    fl_graphics_driver->end_loop();
}
inline void fl_end_polygon() {
    fl_graphics_driver->end_polygon();
}
inline void fl_begin_complex_polygon() {
    fl_graphics_driver->begin_complex_polygon();
}
inline void fl_gap() {
    fl_graphics_driver->gap();
}
```
inline void fl_end_complex_polygon() {
    fl_graphics_driver->end_complex_polygon();
}

// get and use transformed positions:
inline double fl_transform_x(double x, double y) {
    return fl_graphics_driver->transform_x(x, y);
}
inline double fl_transform_y(double x, double y) {
    return fl_graphics_driver->transform_y(x, y);
}
inline double fl_transform_dx(double x, double y) {
    return fl_graphics_driver->transform_dx(x, y);
}
inline double fl_transform_dy(double x, double y) {
    return fl_graphics_driver->transform_dy(x, y);
}
inline void fl_transformed_vertex(double xf, double yf) {
    fl_graphics_driver->transformed_vertex(xf, yf);
}

inline void fl_copy_offscreen(int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy) {
    fl_graphics_driver->copy_offscreen(x, y, w, h, pixmap, srcx, srcy);
}

FL_EXPORT Fl_Offscreen fl_create_offscreen(int w, int h);
FL_EXPORT void fl_begin_offscreen(Fl_Offscreen b);
FL_EXPORT void fl_end_offscreen(void);
FL_EXPORT void fl_delete_offscreen(Fl_Offscreen bitmap);
FL_EXPORT void fl_rescale_offscreen(Fl_Offscreen &ctx);

/* NOTE: doxygen comments here to avoid triplication in os-specific sources */

// Fonts:
/*
Set the current font, which is then used in various drawing routines.
Implemented and documented in src/fl_draw.cxx
*/
FL_EXPORT void fl_font(Fl_Font face, Fl_Fontsize fsize);

inline Fl_Font fl_font() {
    return fl_graphics_driver->font();
}
inline Fl_Fontsize fl_size() {
    return fl_graphics_driver->size();
}

// Information you can get about the current font:
inline int fl_height() {
    return fl_graphics_driver->height();
}
inline int fl_height(int font, int size);
inline int fl_descent() {
    return fl_graphics_driver->descent();
}
FL_EXPORT double fl_width(const char *txt);
inline double fl_width(const char *txt, int n) {
    return fl_graphics_driver->width(txt, n);
}
inline double fl_width(unsigned int c) {
    return fl_graphics_driver->width(c);
}
FL_EXPORT void fl_text_extents(const char *, int &dx, int &dy, int &w, int &h);
inline void fl_text_extents(const char *t, int n, int &dx, int &dy, int &w, int &h) {
    fl_graphics_driver->text_extents(t, n, dx, dy, w, h);
}

FL_EXPORT const char *fl_latin1_to_local(const char *t, int n = -1);
FL_EXPORT const char *fl_local_to_latin1(const char *t, int n = -1);
FL_EXPORT const char *fl_mac_roman_to_local(const char *t, int n = -1);
FL_EXPORT const char *fl_local_to_mac_roman(const char *t, int n = -1);
FL_EXPORT float fl_override_scale();

FL_EXPORT void fl_restore_scale(float s);

FL_EXPORT void fl_draw(const char *str, int x, int y);
FL_EXPORT void fl_draw(int angle, const char *str, int x, int y);
inline void fl_draw(const char *str, int n, int x, int y) {
    fl_graphics_driver->draw(str, n, x, y);
}
inline void fl_draw(int angle, const char *str, int n, int x, int y) {
    fl_graphics_driver->draw(angle, str, n, x, y);
}
inline void fl_rtl_draw(const char *str, int n, int x, int y) {
    fl_graphics_driver->rtl_draw(str, n, x, y);
}

FL_EXPORT void fl_measure(const char *str, int &x, int &y, int draw_symbols = 1);

FL_EXPORT void fl_draw(const char *str, int x, int y, int w, int h, Fl_Align align, Fl_Image *img = 0, int draw_symbols = 1);

void (*callthis)(const char *, int, int, int); Fl_Image *img = 0, int draw_symbols = 1);

FL_EXPORT void fl_frame(const char *s, int x, int y, int w, int h);

FL_EXPORT void fl_frame2(const char *s, int x, int y, int w, int h);

FL_EXPORT void fl_draw_box(Fl_Boxtype, int x, int y, int w, int h, Fl_Color);

FL_EXPORT void fl_draw_check(Fl_Rect bb, Fl_Color col);

void fl_draw_circle(int x, int y, int d, Fl_Color color);

FL_EXPORT void fl_draw_radio(int x, int y, int d, Fl_Color color);

FL_EXPORT void fl_draw_image(const uchar *buf, int X, int Y, int W, int H, int D = 3, int L = 0) {
    fl_graphics_driver->draw_image(buf, X, Y, W, H, D, L);
}

FL_EXPORT void fl_draw_image_mono(const uchar *buf, int X, int Y, int W, int H, int D = 1, int L = 0) {
    fl_graphics_driver->draw_image_mono(buf, X, Y, W, H, D);
}

FL_EXPORT void fl_draw_image(Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D = 3) {
    fl_graphics_driver->draw_image(cb, data, X, Y, W, H, D);
}

FL_EXPORT void fl_draw_image_mono(Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D = 1) {
    fl_graphics_driver->draw_image_mono(cb, data, X, Y, W, H, D);
}

FL_EXPORT uchar *fl_read_image(uchar *p, int X, int Y, int W, int H, int alpha = 0);

FL_EXPORT Fl_RGB_Image *fl_capture_window(Fl_Window *win, int x, int y, int w, int h);

FL_EXPORT int fl_draw_pixmap(const char *const *data, int x, int y, Fl_Color bg = FL_GRAY);

inline int fl_draw_pixmap(/*const*/ char *const *data, int x, int y, Fl_Color bg = FL_GRAY) {
    return fl_draw_pixmap((const char *const *)data, x, y, bg);
}

FL_EXPORT int fl_measure_pixmap(const char *const *data, int &w, int &h);

FL_EXPORT void fl_scroll(int X, int Y, int W, int H, int dx, int dy,
                        void (*draw_area)(void *, int, int, int), void *data);

FL_EXPORT const char *fl_can_do_alpha_blending() {
    return Fl_Graphics_Driver::default_driver()->can_do_alpha_blending();
}

// XForms symbols:
FL_EXPORT int fl_draw_symbol(const char *label, int x, int y, int w, int h, Fl_Color bg = FL_GRAY);

FL_EXPORT int fl_add_symbol(const char *name, void (*drawit)(Fl_Color), int scalable);

FL_EXPORT void fl_set_status(int X, int Y, int W, int H);

FL_EXPORT void fl_set_spot(int font, int size, int X, int Y, int W, int H, Fl_Window *win = 0);

FL_EXPORT void fl_reset_spot(void);

// XForms symbols:
FL_EXPORT int fl_draw_symbol(const char *label, int x, int y, int w, int h, Fl_Color bg = FL_GRAY);

FL_EXPORT int fl_add_symbol(const char *name, void (*drawit)(Fl_Color), int scalable);
13.44 Fl_Export.H

0001 /*
0002 * Windows DLL export.
0003 *
0004 * Copyright 1998-2018 by Bill Spitzak and others.
0005 *
0006 * This library is free software. Distribution and use rights are outlined in
0007 * the file COPYING which should have been included with this file. If this
0008 * file is missing or damaged, see the license at:
0009 *
0010 * https://www.fltk.org/COPYING.php
0011 *
0012 * Please see the following page on how to report bugs and issues:
0013 *
0014 * https://www.fltk.org/bugs.php
0015 */
0016
0017 ifndef Fl_Export_H
0018 # define Fl_Export_H
0019
0020 /*
0021 * The following is used when building DLLs under Windows
0022 * and when building .so's under unix/linux.
0023 */
0024
0025 # ifdef FL_DLL
0026 # ifdef FL_LIBRARY
0027 # define FL_EXPORT __declspec(dllexport)
0028 # else
0029 # define FL_EXPORT __declspec(dllimport)
0030 # endif /* FL_LIBRARY */
0031 # elif __GNUC__ >= 4
0032 # define FL_EXPORT __attribute__ ((visibility ("default")))
0033 # else
0034 # define FL_EXPORT
0035 # endif /* FL_DLL */
0036
0037 #endif /* !Fl_Export_H */

13.45 Fl_File_Browser.H

0001 //
0002 // FileBrowser definitions.
0003 //
0004 // Copyright 1999-2010 by Michael Sweet.
0005 //
0006 // This library is free software. Distribution and use rights are outlined in
0007 // the file COPYING which should have been included with this file. If this
0008 // file is missing or damaged, see the license at:
0009 //
0010 // https://www.fltk.org/COPYING.php
0011 //
0012 // Please see the following page on how to report bugs and issues:
0013 //
0014 // https://www.fltk.org/bugs.php
0015 //
0016 //
0017 /* \file
0018 * Fl_File_Browser widget.
0019 */
0020 //
0021 // Include necessary header files...
0022 //
0023 ifndef _Fl_File_Browser_H_
0024 # define _Fl_File_Browser_H_
0025 # include "Fl_Browser.H"
0026 # include "Fl_File_Icon.H"
0027 # include "filename.H"
0028 # define FL_EXPORT _attribute__ ((visibility ("default")))
0029 # define FL_FILE_BROWSER class...
0030 # define FL_FILE_BROWSER ...
0031 # define FL_FILE_BROWSER ...
0032 //
0033 // Fl_File_Browser class...
0034 //
0035 // FL_EXPORT FL_File_Browser : public Fl_Browser {
0036 //
0037 int filetype;
const char *directory_;  
uchar iconsize_;  
const char *pattern_;  
const char *errmsg_;  

int full_height() const FL_OVERRIDE;  
int item_height(void *) const FL_OVERRIDE;  
int item_width(void *) const FL_OVERRIDE;  
void item_draw(void *, int, int, int, int) const FL_OVERRIDE;  
int incr_height() const FL_OVERRIDE { return (item_height(0) + linespacing()); }  

int filetype() const { return (filetype_); }  
void filetype(int t) { filetype_ = t; }  
void errmsg(const char *emsg);  
const char * errmsg() const { return errmsg_; }  

enum Type { SINGLE = 0, 
          MULTI = 1, 
          CREATE = 2, 
          DIRECTORY = 4 
          }  

Fl_File_Browser(int, int, int, int, const char * = 0);  
~Fl_File_Browser();  

uchar iconsize() const { return (iconsize_); }  
void iconsize(uchar s) { iconsize_ = s; redraw(); }  

void filter(const char *pattern);  
const char * filter() const { return (pattern_); }  

int filetype() const { return (filetype_); }  
void filetype(int t) { filetype_ = t; }  
void errmsg(const char *emsg);  
const char * errmsg() const { return errmsg_; }  


13.46 Fl_File_Chooser.H

// Fl_File_Chooser dialog for the Fast Light Tool Kit (FLTK).  
// Copyright 1998-2024 by Bill Spitzak and others.  
// This library is free software. Distribution and use rights are outlined in  
// the file COPYING which should have been included with this file. If this  
// file is missing or damaged, see the license at:  
// https://www.fltk.org/COPYING.php  
// Please see the following page on how to report bugs and issues:  
// https://www.fltk.org/bugs.php  
//======================================================================  
// DO NOT EDIT FL/Fl_File_Chooser.H and src/Fl_File_Chooser.cxx !!!  
// Please use fluid to change src/Fl_File_Chooser.fl interactively  
// and then use fluid to "write code" or edit and use fluid -c .  
//======================================================================  

#include <FL/Fl.H>  
#include <FL/Fl_Double_Window.H>  
#include <FL/Fl_Group.H>  
#include <FL/Fl_Choice.H>  
#include <FL/Fl_Menu_Button.H>  
#include <FL/Fl_Button.H>  
#include <FL/Fl_Preferences.H>  
#include <FL/Fl_Tile.H>  
#include <FL/Fl_File_Browser.H>  
#include <FL/Fl_Box.H>  
#include <FL/Fl_Check_Button.H>  
#include <FL/Fl_File_Input.H>  
#include <FL/Fl_Return_Button.H>  
#include <FL/fl_ask.H>  

class FL_EXPORT Fl_File_Chooser {
  public:
    enum Type {  
      SINGLE = 0,  
      MULTI = 1,  
      CREATE = 2,  
      DIRECTORY = 4  
    }  

Generated by Doxygen
```cpp
private:
static Fl_Preferences *prefs_;
void (*callback_)(Fl_File_Chooser*, void *);
void *data_;
char directory_[FL_PATH_MAX];
char pattern_[FL_PATH_MAX];
char preview_text_[2048];
type_;  
void favoritesButtonCB();
void favorsCB(Fl_Widget *w);
void fileListCB();
void newdir();
static void previewCB(Fl_File_Chooser *fc);
void showChoiceCB();
void update_favorites();
void update_preview();

public:
Fl_File_Chooser(const char *pathname, const char *pattern, int type_val, const char *title);
Fl_Double_Window *window;
inline void cb_window_i(Fl_Double_Window *, void*);
static void cb_window(Fl_Double_Window *, void*);
Fl_Choice *showChoice;
inline void cb_showChoice_i(Fl_Choice *, void*);
static void cb_showChoice(Fl_Choice *, void*);
Fl_Menu_Button *favoritesButton;
inline void cb_favoritesButton_i(Fl_Menu_Button *, void*);
static void cb_favoritesButton(Fl_Menu_Button *, void*);
Fl_Button *newButton;
private:
inline void cb_newButton_i(Fl_Button *, void*);
static void cb_newButton(Fl_Button *, void*);
inline void cb__i(Fl_Tile *, void*);
static void cb_(Fl_Tile *, void*);
Fl_File_Browser *fileList;
inline void cb_fileList_i(Fl_File_Browser *, void*);
static void cb_fileList(Fl_File_Browser *, void*);
Fl_Return_Button *okButton;
inline void cb_okButton_i(Fl_Return_Button *, void*);
static void cb_okButton(Fl_Return_Button *, void*);
Fl_Button *cancelButton;
inline void cb_cancelButton_i(Fl_Button *, void*);
static void cb_cancelButton(Fl_Button *, void*);
Fl_Double_Window *favWindow;
Fl_File_Browser *favList;
inline void cb_favList_i(Fl_File_Browser *, void*);
static void cb_favList(Fl_File_Browser *, void*);
Fl_Button *favUpButton;
inline void cb_favUpButton_i(Fl_Button *, void*);
static void cb_favUpButton(Fl_Button *, void*);
Fl_Button *favDeleteButton;
inline void cb_favDeleteButton_i(Fl_Button *, void*);
static void cb_favDeleteButton(Fl_Button *, void*);
Fl_Button *favDownButton;
inline void cb_favDownButton_i(Fl_Button *, void*);
static void cb_favDownButton(Fl_Button *, void*);
Fl_Return_Button *favOkButton;
inline void cb_favOkButton_i(Fl_Return_Button *, void*);
static void cb_favOkButton(Fl_Return_Button *, void*);

~Fl_File_Chooser();
void callback(void (*cb)(Fl_File_Chooser *, void *), void *d = 0);
void color(Fl_Color c);
Fl_Color color();
int count();
directory(const char *d);
```
13.47 Fl_File_Icon.H

00143 char * directory();
00144 void filter(const char *p);
00145 const char * filter();
00146 int filter_value();
00147 void filter_value(int f);
00148 void iconsize(uchar s);
00149 uchar iconsize();
00150 void label(const char *l);
00151 const char * label();
00152 void ok_label(const char *l);
00153 const char * ok_label();
00154 void preview(int e);
00155 int preview() const { return previewButton->value(); } }
00156 private:
00157 void showHidden(int e);
00158 void remove_hidden_files();
00159 public:
00160 void rescan();
00161 void rescan_keep_filename();
00162 void show();
00163 void hide();
00164 int shown();
00165 void textcolor(Fl_Color c);
00166 Fl_Color textcolor();
00167 void textfont(Fl_Font f);
00168 Fl_Font textfont();
00169 void textsize(Fl_Fontsize s);
00170 Fl_Fontsize textsize();
00171 void type(int t);
00172 int type();
00173 void * user_data() const;
00174 void user_data(void *d);
00175 const char *value(int f = 1);
00176 void value(const char *filename);
00177 int visible();
00178 void position(int x, int y);
00179 int x() const;
00180 int y() const;
00181 int h() const;
00182 int w() const;
00183 size(int x, int y, int w, int h);
00184 resize(int x, int y, int w, int h);
00185 static const char *add_favorites_label;
00186 static const char *all_files_label;
00187 static const char *custom_filter_label;
00188 static const char *existing_file_label;
00189 static const char *favorites_label;
00190 static const char *filename_label;
00191 static const char *filesystems_label;
00192 static const char *manage_favorites_label;
00193 static const char *new_directory_label;
00194 static const char *new_directory_tooltip;
00195 static const char *preview_label;
00196 static const char *save_label;
00197 static const char *show_label;
00198 static const char *hidden_label;
00199 static Fl_File_Sort_F *sort;
00200 private:
00201 Fl_Widget * ext_group;
00202 Fl_Widget * add_extra(Fl_Widget* gr);
00203 protected:
00204 void show_error_box(int val);
00205 void show_error_box(const char *msg, int val);
00206 FL_EXPORT char *fl_dir_chooser(const char *message,const char *fname,int relative=0);
00207 FL_EXPORT char *fl_file_chooser(const char *message,const char *pat,const char *fname,int relative=0);
00208 FL_EXPORT void fl_file_chooser_callback(void (*cb)(const char*));
00209 FL_EXPORT void fl_file_chooser_ok_label(const char*l);
00210 #endif

13.47 Fl_File_Icon.H

00001 //
00002 // Fl_File_Icon definitions.
00003 //
00004 Copyright 1999-2010 by Michael Sweet.
00005 //
00006 This library is free software. Distribution and use rights are outlined in
00007 the file "COPYING" which should have been included with this file. If this
00008 file is missing or damaged, see the license at:
00009 //
00010 // https://www.fltk.org/COPYING.php
00011 //
00012 Please see the following page on how to report bugs and issues:
00013 //

Generated by Doxygen
```cpp
#ifndef _Fl_Fl_File_Icon_H_
#define _Fl_Fl_File_Icon_H_

#include "Fl.H"

class Fl_EXPORT Fl_File_Icon {
  static Fl_File_Icon *first_;  // Pointer to first icon/filetype
  Fl_File_Icon *next_;  // Pointer to next icon/filetype
  const char *pattern_;  // Pattern string
  int type_;  // Match only if directory or file?
  int num_data_;  // Number of data elements
  int alloc_data_;  // Number of allocated elements
  short *data_;  // Icon data

public:
  Fl_File_Icon(const char *p, int t, int nd = 0, short *d = 0);
  ~Fl_File_Icon();

  short *add(short d);
  short *add_color(Fl_Color c) {
    short *d = add((short)COLOR); add({short}{c >> 16}); add({short}c); return (d); }
  short *add_vertex(int x, int y)
  {
    short *d = add((short)VERTEX); add({short}x); add({short}y); return (d); }
  short *add_vertex(float x, float y)
  {
    short *d = add((short)VERTEX); add({short}(x * 10000.0));
    add({short}(y * 10000.0)); return (d); }
  void clear() { num_data_ = 0; }
  void draw(int x, int y, int w, int h, Fl_Color ic, int active = 1);
  static void labeltype(const Fl_Label *o, int x, int y, int w, int h, Fl_Align a);
  int load(const char *f);
  int load_fti(const char *fti);
  int load_image(const char *i);
  Fl_File_Icon *next() { return (next_); }
  const char *pattern() { return (pattern_); }
};
```
```cpp
00126 int size() { return (num_data_); }
00129 int type() { return (type_); }
00142 short *value() { return (data_); }
00145 static Fl_File_Icon *find(const char *filename, int filetype = ANY);
00149 static Fl_File_Icon *first() { return (first_); }
00151 static void load_system_icons(void);
00152
00153 #endif // !_Fl_Fl_File_Icon_H_

13.48 Fl_File_Input.H

00001 //
00002 // File_Input header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2022 by Bill Spitzak and others.
00006 //
00007 // This library is free software. Distribution and use rights are outlined in
00008 // the file "COPYING" which should have been included with this file. If this
00009 // file is missing or damaged, see the license at:
00010 //
00011 // https://www.fltk.org/COPYING.php
00012 //
00013 // Please see the following page on how to report bugs and issues:
00014 //
00015 // https://www.fltk.org/bugs.php
00016 //
00017 #ifndef Fl_File_Input_H
00018 #define Fl_File_Input_H
00019
00020 #include <FL/Fl_Input.H>
00021
00024 class FL_EXPORT Fl_File_Input : public Fl_Input {
00025     char ok_entry_;
00026     uchar down_box_;
00029     short buttons_[200];
00030     short pressed_;
00032     void draw_buttons();
00033     int handle_button(int event);
00034     void update_buttons();
00035
00037 public:
00038     Fl_File_Input(int X, int Y, int W, int H, const char *L=0);
00039     int handle(int event) FL_OVERRIDE;
00040     protected:
00041     void draw() FL_OVERRIDE;
00042
00044 public:
00047     Fl_Boxtype down_box() const { return (Fl_Boxtype)down_box_; }
00049     void draw_box(Fl_Boxtype b) { down_box_ = b; }
00050     Fl_Color errorcolor() const { return FL_RED; }  
00051     void errorcolor(Fl_Color c) { [void];c; }
00052     int value(const char *str);
00053     int value(const char *str, int len);
00054     const char *value() { return Fl_Input_::value(); }
00055
00057 #endif // !Fl_File_Input_H

13.49 Fl_Fill_Dial.H

00001 //
00002 // Filled dial header file for the Fast Light Tool Kit (FLTK).
```
#ifndef Fl_Fill_Dial_H
#define Fl_Fill_Dial_H

#include "Fl_Dial.H"

class FL_EXPORT Fl_Fill_Dial : public Fl_Dial {
 public:
 Fl_Fill_Dial(int X, int Y, int W, int H, const char *L);
};

#endif

#ifndef Fl_Fill_Slider_H
#define Fl_Fill_Slider_H

#include "Fl_Slider.H"

class FL_EXPORT Fl_Fill_Slider : public Fl_Slider {
 public:
 Fl_Fill_Slider(int X, int Y, int W, int H, const char *L=0);

#endif

#ifndef Fl_Fill_Slider_H
#define Fl_Fill_Slider_H

#include "Fl_Slider.H"

class FL_EXPORT Fl_Fill_Slider : public Fl_Slider {
 public:
 Fl_Fill_Slider(int X, int Y, int W, int H, const char *L=0);

#endif

#ifndef Fl_Flex_H
#define Fl_Flex_H

#endif

#ifndef Fl_Flex_H
#define Fl_Flex_H

#endif

#ifndef Fl_Flex_H
#define Fl_Flex_H

#endif

#ifndef Fl_Flex_H
#define Fl_Flex_H

#endif

#ifndef Fl_Flex_H
#define Fl_Flex_H

#endif

#ifndef Fl_Flex_H
#define Fl_Flex_H

#endif

#ifndef Fl_Flex_H
#define Fl_Flex_H

#endif

#ifndef Fl_Flex_H
#define Fl_Flex_H

#endif

#ifndef Fl_Flex_H
#define Fl_Flex_H

#endif
```cpp
#include <FL/Fl_Group.H>
class FL_EXPORT Fl_Flex : public Fl_Group {
  int margin_left_; // left margin
  int margin_top_; // top margin
  int margin_right_; // right margin
  int margin_bottom_; // bottom margin
  int gap_; // gap between widgets
  int fixed_size_size_; // number of fixed size widgets in array
  int fixed_size_alloc_; // allocated size of fixed size array
  Fl_Widget **fixed_size_; // array of fixed size widgets
  bool need_layout_; // true if layout needs to be calculated

public:
  enum { // values for type(int)
    VERTICAL = 0,
    HORIZONTAL = 1,
    COLUMN = 0,
    ROW = 1
  };

  // FLTK standard constructor
  Fl_Flex(int X, int Y, int W, int H, const char *L = 0);

  // original Fl_Flex constructors:
  // backwards compatible if direction enums { ROW | COLUMN } are used
  Fl_Flex(int direction);
  Fl_Flex(int w, int h, int direction);
  Fl_Flex(int x, int y, int w, int h, int direction);
  virtual ~Fl_Flex();

  void fixed(Fl_Widget &w, int size) {
    fixed(&w, size);
  }

  void fixed(Fl_Widget *w, int size);
  int fixed(Fl_Widget *w) const;

protected:
  void init(int t = VERTICAL);
  virtual int alloc_size(int size) const;
  void on_remove(int) FL_OVERRIDE;
  void draw() FL_OVERRIDE;

public:
  void need_layout(int set) {
    if (set) need_layout_ = true;
    else need_layout_ = false;
  }

  bool need_layout() const {
    return need_layout_;}

  int margin() const { return margin_left_; }

  int margin(int *left, int *top, int *right, int *bottom) const {
    if (left) *left = margin_left_; 
    if (top) *top = margin_top_; 
    if (right) *right = margin_right_; 
    if (bottom) *bottom = margin_bottom_; 
    if (margin_left_ == margin_top_ && margin_top_ == margin_right_ && margin_right_ == margin_bottom_)
      return 1;
    return 0;
  }

  void margin(int m, int g = -1) {
    if (m < 0)
      m = 0;
    margin_left_ = margin_top_ = margin_right_ = margin_bottom_ = m;
    if (g >= 0)
      gap_ = g;
    need_layout(1);
  }
};
```
void margin(int left, int top, int right, int bottom) {
  margin_left_ = left < 0 ? 0 : left;
  margin_top_ = top < 0 ? 0 : top;
  margin_right_ = right < 0 ? 0 : right;
  margin_bottom_ = bottom < 0 ? 0 : bottom;
  need_layout(1);
}

int gap() const {
  return gap_;}

void gap(int g) {
  gap_ = g < 0 ? 0 : g;
  need_layout(1);
}

int horizontal() const {
  return type() == Fl_Flex::HORIZONTAL ? 1 : 0;
}

// Calculate the layout of the widget and redraw it.
void layout();

int spacing() const {
  return gap_;}

void spacing(int i) {
  gap(i);
  need_layout(1);
}

};

#endif // Fl_Flex_H

13.52 Fl_Float_Input.H

#include "Fl_Input.H"

class FL_EXPORT Fl_Float_Input : public Fl_Input {
public:
  Fl_Float_Input(int X,int Y,int W,int H,const char *l = 0);
};

13.53 Fl_FormsBitmap.H
Forms pixmap header file for the Fast Light Tool Kit (FLTK).

Copyright 1998-2010 by Bill Spitzak and others.

This library is free software. Distribution and use rights are outlined in the file "COPYING" which should have been included with this file. If this file is missing or damaged, see the license at:

https://www.fltk.org/COPYING.php

Please see the following page on how to report bugs and issues:

https://www.fltk.org/bugs.php

class FL_EXPORT Fl_FormsPixmap : public Fl_Widget {
  Fl_Pixmap *b;
  protected:
  void draw() FL_OVERRIDE;
  public:
  Fl_FormsPixmap(Fl_Boxtype t, int X, int Y, int W, int H, const char *L= 0);
  void set(const char * const * bits);
  void Pixmap(Fl_Pixmap *B) {b = B;}
  Fl_Pixmap *Pixmap() const {return b;
}

#endif
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

file

/* file

 Fl_Free widget . */

#ifndef Fl_Free_H
#define Fl_Free_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

#define FL_NORMAL_FREE 1
#define FL_SLEEPING_FREE 2
#define FL_INPUT_FREE 3
#define FL_CONTINUOUS_FREE 4
#define FL_ALL_FREE 5

typedef int (*FL_HANDLEPTR)(Fl_Widget *, int , float, float, char);

class FL_EXPORT Fl_Free : public Fl_Widget {
  FL_HANDLEPTR hfunc;

  static void step(void *);

  protected:

  void draw() FL_OVERRIDE;

  public:

  int handle(int e) FL_OVERRIDE;

  Fl_Free(uchar t, int X, int Y, int W, int H, const char *L, FL_HANDLEPTR hdl);

  ~Fl_Free();

};

#endif

// old event names for compatibility:
#define FL_MOUSE FL_DRAG
#define FL_DRAW 100
#define FL_STEP 101
#define FL_FREEMEM 102
#define FL_FREEZE 103
#define FL_THAW 104

13.56  Fl_GIF_Image.H

GIF image header file for the Fast Light Tool Kit (FLTK).

Copyright 1998-2023 by Bill Spitzak and others.

This library is free software. Distribution and use rights are outlined in
the file "COPYING" which should have been included with this file. If this
file is missing or damaged, see the license at:

https://www.fltk.org/COPYING.php

Please see the following page on how to report bugs and issues:

https://www.fltk.org/bugs.php

file

/* file

 Fl_GIF_Image widget . */

#ifndef Fl_GIF_Image_H
#define Fl_GIF_Image_H

#include "Fl_Pixmap.H"

class FL_EXPORT Fl_GIF_Image : public Fl_Pixmap {

  public:

  Fl_GIF_Image(const char * filename);

  Fl_GIF_Image(const char * imagename, const unsigned char *data);

  Fl_GIF_Image(const char * imagename, const unsigned char *data, const size_t length);

  static bool is_animated(const char *name_);

  static bool animate;

  protected:

  bool is_animated(const char *name_);

  static bool animate;

};

protected:

// Protected constructors needed for animated GIF support through Fl_Anim_GIF_Image.

Fl_GIF_Image(const char * filename, bool anim);
Fl_GIF_Image(const char * imagename, const unsigned char *data, const size_t length, bool anim);

// Protected default constructor needed for Fl_Anim_GIF_Image.
Fl_GIF_Image();

void load_gif_(class Fl_Image_Reader &rdr, bool anim=false);

void load(const char * filename, bool anim);
void load(const char * imagename, const unsigned char *data, const size_t length, bool anim);

// Internal structure to "glue" animated GIF support into Fl_GIF_Image.
struct GIF_FRAME {
  int ifrm, width, height, x, y, w, h,
  clrs, bkgd, trans,
  dispose, delay;
  const uchar *bptr;
  const struct CPAL {
    uchar r, g, b;
  } *cpal;
  GIF_FRAME(int frame, uchar *data) : ifrm(frame), bptr(data) {}
  GIF_FRAME(int frame, int W, int H, int fx, int fy, int fw, int fh, uchar *data) :
    ifrm(frame), width(W), height(H), x(fx), y(fy), w(fw), h(fh), bptr(data) {
    void disposal(int mode, int delay) { dispose = mode; this->delay = delay; }
    void colors(int nclrs, int bg, int tp) { clrs = nclrs; bkgd = bg; trans = tp; }
    }
  void disposal(int mode, int delay) { dispose = mode; this->delay = delay; }
  void colors(int nclrs, int bg, int tp) { clrs = nclrs; bkgd = bg; trans = tp; }
  void disposal(int mode, int delay) { dispose = mode; this->delay = delay; }
  void colors(int nclrs, int bg, int tp) { clrs = nclrs; bkgd = bg; trans = tp; }
  void disposal(int mode, int delay) { dispose = mode; this->delay = delay; }
  void colors(int nclrs, int bg, int tp) { clrs = nclrs; bkgd = bg; trans = tp; }
};

// Internal virtual methods, which are called during decoding to pass data
// to the Fl_Anim_GIF_Image class.
virtual void on_frame_data(GIF_FRAME &);
virtual void on_extension_data(GIF_FRAME &);

private:

void lzw_decode(Fl_Image_Reader &rdr, uchar *Image, int Width, int Height, int CodeSize, int
ColorMapSize, int Interlace);

#ifdef

Fl_GI_Window widget . */
#endif

Fl_Gl_Choice; // structure to hold result of glXChooseVisual
Fl_Gl_Window_Driver;
class FL_EXPORT Fl_Gl_Window : public Fl_Window {
friend class Fl_Gl_Window_Driver;
Fl_Gl_Window_Driver *pG1WindowDriver;

int mode_; int *alist;
Fl_GI_Choice *g;
GLContext context_; char valid_f_; char damage1_; // damage() of back buffer
virtual void draw_overlay();
void init();

void *overlay;

static int can_do(int, const int *);
int mode(int, const int *); static int gl_plugin_linkage();

Generated by Doxygen
protected:
void draw_begin();
void draw() FL_OVERRIDE;
void draw_end();

public:
void show() FL_OVERRIDE;
void show(int a, char **b) {Fl_Window::show(a,b);}
void flush() FL_OVERRIDE;
void hide() FL_OVERRIDE;
void resize(int,int,int,int) FL_OVERRIDE;
int handle(int) FL_OVERRIDE;

char valid() const {return valid_f_ & 1;}
void valid(char v) {if (v) valid_f_ |= 1; else valid_f_ &= 0xfe;}
void invalidate();

char context_valid() const {return valid_f_ & 2;}
void context_valid(char v) {if (v) valid_f_ |= 2; else valid_f_ &= 0xfd;}

static int can_do(int m) {return can_do(m,0);}
static int can_do(const int *m) {return can_do(0, m);}
int can_do() {return can_do(mode_,alist);}
int mode(int a) {return mode(a,0);}
int mode(const int *a) {return mode(0, a);}
GLContext context() const {return context_;}
void context(GLContext, int destroy_flag = 0);
void make_current();
void swap_buffers();
void swap_interval();
void ortho();

~Fl_Gl_Window();

Generated by Doxygen

typedef void (*Fl_Draw_Image_Cb)(void* data, int x, int y, int w, uchar* buf);

virtual void draw_pixmap(Fl_Pixmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy);

The graphics driver can implement up to 6 virtual member functions to draw images:

- Fl_Graphics_Driver::gl_start()
- Fl_Graphics_Driver::fl_convert_pixmap(const char *const* cdata, uchar* out, Fl_Color bg)
- Fl_Graphics_Driver::fl_copy_offscreen(int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy);
- Fl_Graphics_Driver::fl_draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D);
- Fl_Graphics_Driver::fl_draw_image_mono(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D);
- Fl_Graphics_Driver::fl_draw_image(const uchar * buf, int X, int Y, int W, int H, int D, int L);
- Fl_Graphics_Driver::fl_draw_image_mono(const uchar * buf, int X, int Y, int W, int H, int D, int L);

- Fl_SVG_Image
- Fl_Pixmap
- Fl_Bitmap
- Fl_RGB_Image

virtual void draw_rgb(Fl_RGB_Image *,....).

- Consider an image object, say from class Fl_RGB_Image. Fl_RGB_Image::draw() takes care of resizing and caching the Pixmap to the adequate drawing size. If Fl_XXX_Graphics_Driver does not re-implement this function, Fl_Graphics_Driver::draw_rgb(Fl_RGB_Image *,....) runs. It internally calls Fl_XXX_Graphics_Driver::draw_fixed(Fl_RGB_Image *,....) which draws the image from its cached form which already has the adequate size.

- Some drivers implement, for a given image class, the function of both groups, e.g. : Fl_Quartz_Graphics_Driver.

Each resizes the image, caches it, and calls the platform-specific implementation of draw_fixed(image-class,....,....) with the cached image.

- Only draw_fixed() as in these examples:
- Fl_GDI_Graphics_Driver::draw_rgb() which runs Fl_GDI_Graphics_Driver::draw_fixed(Fl_RGB_Image *,....) because scale-and-draw may require function Alphablend() that draws the image from its cached form which already has the adequate size.

- Some drivers implement, for a given image class, only the function of the 1st group or only draw_fixed() as in these examples:
- Fl_Xlib_Graphics_Driver implements only draw_fixed(Fl_RGB_Image *,....) as in these examples:
- Fl_GDI_Graphics_Driver::draw_fixed(Fl_RGB_Image *,....) runs. It internally calls Fl_XXX_Graphics_Driver::draw_fixed(Fl_RGB_Image *,....) that draws the image from its cached form which already has the adequate size.

- Some drivers implement, for a given image class, only the function of each group.

- Fl_Quartz_Graphics_Driver implements only draw_rgb(Fl_RGB_Image *,....) as in these examples:
- Fl_Xlib_Graphics_Driver implements only draw_fixed(Fl_RGB_Image *,....). The library takes care of resizing and caching the Pixmap to the adequate drawing size.

- Fl_Quartz_Graphics_Driver::draw_rgb() which runs Fl_Quartz_Graphics_Driver::draw_fixed(Fl_RGB_Image *,....) because function FlAlphaBlend requires the RGB image. If Fl_XXX_Graphics_Driver does not re-implement this function, this code runs and is expected to draw the image adequately scaled to its drawing size. If Fl_XXX_Graphics_Driver does not re-implement this function, Fl_Graphics_Driver::draw_rgb(Fl_RGB_Image *,....) runs. It internally calls Fl_XXX_Graphics_Driver::draw_rgb(Fl_RGB_Image *,....) that draws the image from its cached form which already has the adequate size.

- Some drivers implement, for a given image class, only the function of the 1st group or only draw_fixed() as in these examples:
- Fl_GDI_Graphics_Driver::draw_rgb() which runs Fl_GDI_Graphics_Driver::draw_fixed(Fl_RGB_Image *,....) as in these examples:
- Fl_Xlib_Graphics_Driver implements only draw_fixed(Fl_RGB_Image *,....) as in these examples:
- Fl_Quartz_Graphics_Driver.

- Some drivers implement, for a given image class, only the function of the 1st group or only draw_fixed() as in these examples:
- Fl_GDI_Graphics_Driver::draw_rgb() which runs Fl_GDI_Graphics_Driver::draw_fixed(Fl_RGB_Image *,....) as in these examples:
- Fl_Xlib_Graphics_Driver implements only draw_fixed(Fl_RGB_Image *,....) as in these examples:
- Fl_Quartz_Graphics_Driver.

- Some drivers implement, for a given image class, only the function of the 1st group or only draw_fixed() as in these examples:
static const matrix m0;
Fl_Font font_;
Fl_Fontsize size_; 
Fl_Color color_; 
int sptr;
static const int matrix_stack_size = FL_MATRIX_STACK_SIZE;
matrix stack[FL_MATRIX_STACK_SIZE]; 
int n;
int gap_; 
enum SHAPE {NONE=0, LINE, LOOP, POLYGON, POINTS, COMPLEX_POLYGON} what;
rstackptr;
static const int region_stack_max = FL_REGION_STACK_SIZE - 1;
Fl_Region rstack[FL_REGION_STACK_SIZE];
Fl_Font_Descriptor *font_descriptor_; 
p_size;
typedef struct { float x; float y; } XPOINT;
xpoint;
virtual void global_gc();
virtual void cache(Fl_Pixmap *img);
virtual void cache(Fl_Bitmap *img);
virtual void cache(Fl_RGB_Image *img);
virtual void uncache(Fl_RGB_Image *img, fl_uintptr_t &id_, fl_uintptr_t &mask_);

// --- implementation is in src/drivers/xxx/Fl_xxx_Graphics_Driver_image.cxx
virtual void draw_image(const uchar * buf, int X,int Y,int W,int H, int D=3, int L=0);
virtual void draw_image_mono(const uchar * buf, int X,int Y,int W,int H, int D=1, int L=0);
virtual void draw_image_mono(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D=3);
virtual void draw_imagemono(Fl_Draw_image_Cb cb, void* data, int X, int Y, int W, int H, int D=1);
virtual void draw_rgb(Fl_RGB_Image *rgb, int XP, int YP, int WP, int HP, int cx, int cy);
virtual void draw_pixmap(Fl_Pixmap *pm, int XP, int YP, int WP, int HP, int cx, int cy);
virtual void draw_bitmap(Fl_Bitmap *bm, int XP, int YP, int WP, int HP, int cx, int cy);
virtual void copy_offscreen(int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy);
static void change_image_size(Fl_Image *img, int W, int H) {
  img->w(W);
  img->h(H);
}

// Support function for image drawing
virtual void uncache_pixmap(fl_uintptr_t p);
// accessor functions to protected image members
int start_image(Fl_Image *img, int XP, int YP, int WP, int HP, int &cx, int &cy, int &X, int &Y, int &W, int &H);
static fl_uintptr_t * id(Fl_RGB_Image *rgb) {return &(rgb->id_);}
static fl_uintptr_t * id(Fl_Pixmap *pm) {return &(pm->id_);}
static fl_uintptr_t * id(Fl_Bitmap *bm) {return &(bm->id_);}
static fl_uintptr_t * mask(Fl_RGB_Image *rgb) {return &(rgb->mask_);}
static fl_uintptr_t * mask(Fl_Pixmap *pm) {return &(pm->mask_);}
static void cache_w_h(Fl_Pixmap *pm, int *&pwidth, int *&pheight) {
pwidth = &(pm->cache_w_);
pheight = &(pm->cache_h_);
}
static void cache_w_h(Fl_Bitmap *bm, int *&pwidth, int *&pheight) {
pwidth = &(bm->cache_w_);
pheight = &(bm->cache_h_);
}
static void cache_w_h(Fl_RGB_Image *rgb, int *&pwidth, int *&pheight) {
pwidth = &(rgb->cache_w_);
pheight = &(rgb->cache_h_);
}
static Fl_Offscreen get_offscreen_and_delete_image_surface(Fl_Image_Surface *);
static void draw_empty(Fl_Image *img, int X, int Y) {img->draw_empty(X, Y);
}
static unsigned need_pixmap_bg_color;

public:
virtual -Fl_Graphics_Driver();
static Fl_Graphics_Driver &default_driver();

// support of "complex shapes"
void push_matrix();
void pop_matrix();
void load_identity();
void load_matrix(double a, double b, double c, double d, double x, double y);
void mult_matrix(double a, double b, double c, double d, double x, double y);
void rotate(double d);
void translate(double x, double y);
double transform_x(double x, double y);
double transform_y(double x, double y);
double transform_dx(double x, double y);
double transform_dy(double x, double y);
inline Fl_Font_Descriptor *font_descriptor() { return font_descriptor_;

void font_descriptor(Fl_Font_Descriptor *d) { font_descriptor_ = d;}

virtual void scale(float s);
virtual void scale(float s);
virtual void draw_image_mono(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D);
virtual void uncache(Fl_RGB_Image *img, fl_uintptr_t &id_, fl_uintptr_t &mask_);
virtual void draw_empty(Fl_Image *img, int X, int Y);
virtual void draw_image(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D);
virtual void draw_image(Fl_Draw_image_Cb cb, void* data, int X, int Y, int W, int H, int D);
virtual void draw_rgb(Fl_RGB_Image *rgb, int XP, int YP, int WP, int HP, int cx, int cy);
virtual void draw_pixmap(Fl_Pixmap *pm, int XP, int YP, int WP, int HP, int cx, int cy);
virtual void draw_bitmap(Fl_Bitmap *bm, int XP, int YP, int WP, int HP, int cx, int cy);
virtual void copy_offscreen(int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy);
static void change_image_size(Fl_Image *img, int W, int H) {img->w(W);
  img->h(H);
}
virtual void rect(int x, int y, int w, int h);
virtual void focus_rect(int x, int y, int w, int h, int r);
virtual void rectf(int x, int y, int w, int h);
virtual void _rbox(int fill, int x, int y, int w, int h, int r);
virtual void rounded_rect(int x, int y, int w, int h, uchar r, uchar g, uchar b);
virtual void rounded_rectf(int x, int y, int w, int h, int r);
virtual void colored_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b);
virtual void line(int x, int y, int x1, int y1);
virtual void line(int x, int y, int x1, int y1, int x2, int y2);
virtual void xyline(int x, int y, int x1);
virtual void xyline(int x, int y, int x1, int y2);
virtual void xyline(int x, int y, int x1, int y2, int x3);
virtual void yxline(int x, int y, int y1);
virtual void yxline(int x, int y, int y1, int x2);
virtual void yxline(int x, int y, int y1, int x2, int y3);
virtual void loop(int x0, int y0, int x1, int y1, int x2, int y2);
virtual void loop(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3);
virtual void polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3);
virtual void polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3);
virtual void push_clip(int x, int y, int w, int h);
virtual int clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H);
virtual int not_clipped(int x, int y, int w, int h);
virtual void push_no_clip();
virtual void pop_clip();
virtual Fl_Region clip_region();
virtual void clip_region(Fl_Region r);
virtual void restore_clip();
virtual void begin_points();
virtual void begin_line();
virtual void begin_loop();
virtual void begin_polygon();
virtual void begin_complex_polygon();
virtual void transformed_vertex(double xf, double yf);
virtual void transformed_vertex0(float x, float y);
virtual void vertex(double x, double y);
virtual void end_points();
virtual void end_line();
virtual void end_loop();
virtual void fixloop();
virtual void end_polygon();
virtual void end_complex_polygon();
virtual bool can_fill_non_convex_polygon() { return true; }
virtual void push_no_clip();
virtual void pop_clip();
virtual Fl_Region clip_region();
virtual void clip_region(Fl_Region r);
virtual void restore_clip();
virtual void draw_circle(int x, int y, int d, Fl_Color c);
virtual void curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3);
virtual void line_style(int style, int width=0, char * dashes=0);
virtual void color(Fl_Color c);
virtual void set_color(Fl_Color i, unsigned int c);
virtual void free_color(Fl_Color i, int overlay);
virtual Fl_Color color();
virtual void draw(const char * str, int nChars, int x, int y);
virtual void draw(const char * str, int nChars, float x, float y);
virtual void draw(int angle, const char * str, int nChars, int x, int y);
virtual void rtl_draw(const char * str, int nChars, int x, int y);
virtual int has_feature(driver_feature feature);
virtual void font(Fl_Font face, Fl_Fontsize fsize);
virtual Fl_Font font();
virtual Fl_Fontsize size();
virtual double width(const char *, int nChars, int x, int y);
virtual double width(unsigned int c);
virtual void text_extents(const char *, int n, int& dx, int& dy, int& w, int& h);
virtual int height();
virtual int descent();
virtual void gc(int i);
virtual void gc(void);
virtual uchar **mask_bitmap();
virtual float scale_font_for_PostScript(Fl_Font_Descriptor *desc, int s);
virtual float scale_bitmap_for_PostScript();
virtual void add_rectangle_to_region(Fl_Region r, int x, int y, int w, int h);
virtual Fl_Region XRectangleRegion(int x, int y, int w, int h);
virtual void XDestroyRegion(Fl_Region r);
virtual const char * get_font_name(Fl_Font fnum, int * ap);
virtual int get_font_sizes(Fl_Font fnum, int * sizep);
virtual Fl_Font set_fonts(const char *name);
virtual Fl_Fontdesc* calc_fl_fonts(void);
virtual unsigned font_desc_size();
virtual const char *font_name(int num);
virtual void font_name(int num, const char *name);
// Default implementation may be enough
virtual void overlay_rect(int x, int y, int w, int h);
virtual float override_scale();
virtual void restore_scale(float);
virtual PangoFontDescription* pango_font_description() { return NULL; }
virtual void antialias(int state);
virtual int antialias();
virtual void delete_bitmask(fl_uintptr_t bm);

#ifndef FL_DOXYGEN

/* This class is not part of FLTK’s public API. Platforms usually define a derived class called Fl_XXX_Font_Descriptor containing extra platform-specific data/functions. This is a class for an actual system font, with junk to help choose it and info on character sizes. Each Fl_Fontdesc has a linked list of these. These are created the first time each system font/size combination is used. */

class Fl_Font_Descriptor {
public:
  Fl_Font_Descriptor *next;
  Fl_Fontsize size;
  Fl_Font_Descriptor(const char * fontname, Fl_Fontsize size);
  virtual FL_EXPORT ~Fl_Font_Descriptor() {}
  int ascent, descent;
  unsigned int listbase; // base of display list, 0 = none
};

template <typename Derived>

// This struct is not part of FLTK’s public API.

struct Fl_Fontdesc {
const char *name;
char fontname[128]; // "Pretty" font name
Fl_Font_Descriptor *first; // linked list of sizes of this style
};

/* Abstract class Fl_Scalable_Graphics_Driver is platform-independent. It supports the scaling of all graphics coordinates by a float factor helpful to support HiDPI displays. This class does:
- compute scaled coordinates
- scale the cached offscreen of image objects
- scale the pixel arrays used when performing direct image draws
- call the member functions of a platform-specific, Fl_Scalable_Graphics_Driver-derived class that do the drawings with adequately scaled coordinates. The member functions are named with the _unscaled suffix.
- scale and unscale the clipping region.

This class is presently used by the X11 and Windows platforms to support HiDPI displays. In the future, it may also be used by other platforms. */

class FL_EXPORT Fl_Scalable_Graphics_Driver : public Fl_Graphics_Driver {
public:
  Fl_Scalable_Graphics_Driver();
  // This function aims to compute accurately int(x * s) in presence of rounding errors existing with floating point numbers
  // and that sometimes differ between 32 and 64 bits.
  static inline int floor(int x, float s) { return int(x * s + 0.001f); }
  inline int floor(int x) { return Fl_Scalable_Graphics_Driver::floor(x, scale()); }
protected:
  int line_width_;
  virtual Fl_Region scale_clip(float r);
  void unscale_clip(Fl_Region r);
  void point(int x, int y) FL_OVERRIDE;
  virtual void point_unscaled(float x, float y);
  void rect(int x, int y, int w, int h) FL_OVERRIDE;
  void rectf(int x, int y, int w, int h) FL_OVERRIDE;
  virtual void rectf_unscaled(int x, int y, int w, int h);
  void line(int x, int y, int x1, int y1) FL_OVERRIDE;
  virtual void line_unscaled(int x, int y, int x1, int y1);
  virtual void line_unscaled(int x, int y, int x1, int y1, int x2, int y2);
  virtual void xyline(int x, int y, int y1, int x3) FL_OVERRIDE;
  virtual void xyline_unscaled(int x, int y, int y1, int x3);
  virtual void yxline(int x, int y, int x1) FL_OVERRIDE;
  virtual void yxline_unscaled(int x, int y, int x1, int y2);
void loop(int x0, int y0, int x1, int y1, int x2, int y2); FL_OVERRIDE;
void loop(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3); FL_OVERRIDE;
void polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3); FL_OVERRIDE;
void polygon_unscaled(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3);
void circle(double x, double y, double r) FL_OVERRIDE;
void ellipse_unscaled(double xt, double yt, double rx, double ry);
void font(Fl_Font face, Fl_Fontsize size) FL_OVERRIDE;
Fl_Font font() FL_OVERRIDE;
void text_extents(const char *str, int n, int &dx, int &dy, int &w, int &h) FL_OVERRIDE;
void text_extents_unscaled(const char *str, int n, int &dx, int &dy, int &w, int &h);
int height() FL_OVERRIDE;
int descent() FL_OVERRIDE;
void draw(const char *str, int n, int x, int y) FL_OVERRIDE;
void draw_unscaled(const char *str, int n, int x, int y);
void rtl_draw(const char *str, int n, int x, int y) FL_OVERRIDE;
void rtl_draw_unscaled(const char *str, int n, int x, int y);
void draw(int angle, const char *str, int n, int x, int y) FL_OVERRIDE;
void draw_unscaled(int angle, const char *str, int n, int x, int y);
void draw(const char *str, int nChars, float x, float y) FL_OVERRIDE;
void draw_mono_unscaled(const uchar *buf, int x, int y, int w, int h, int d, int l);
void draw_image_rescale(void *buf, Fl_Draw_Image_Cb cb, int X, int Y, int W, int H, int D, int L, 
  bool mono); FL_OVERRIDE;
void draw_image_unscaled(const uchar *buf, int X, int Y, int W, int H, int D, int L, 
  bool mono); FL_OVERRIDE;
void draw_image(const uchar *buf, int X, int Y, int W, int H, int D, int L); FL_OVERRIDE;
void draw_image_unscaled(const uchar *buf, int X, int Y, int W, int H, int D, int L, 
  bool mono); FL_OVERRIDE;
void draw_image(const uchar *buf, int X, int Y, int W, int H, int D, int L); FL_OVERRIDE;
void draw_image_unscaled(const uchar *buf, int X, int Y, int W, int H, int D, int L); FL_OVERRIDE;
void draw_image_mono(const uchar *buf, int X, int Y, int W, int H, int D); FL_OVERRIDE;
void draw_image_mono_unscaled(const uchar *buf, int X, int Y, int W, int H, int D, int L); FL_OVERRIDE;
void draw_image_mono(const uchar *buf, int X, int Y, int W, int H, int D); FL_OVERRIDE;


class Fl_Grid::Cell

class Fl_Grid

Fl_Grid is a container (layout) widget with multiple columns and rows.
Typedefs

- typedef unsigned short Fl_Grid_Align
  
  Fl_Grid type for child widget alignment control.

Variables

- const Fl_Grid_Align FL_GRID_BOTTOM = 0x0002
  
  Align the widget at the bottom of the cell.

- const Fl_Grid_Align FL_GRID_BOTTOM_LEFT = FL_GRID_BOTTOM | FL_GRID_LEFT
- const Fl_Grid_Align FL_GRID_BOTTOM_RIGHT = FL_GRID_BOTTOM | FL_GRID_RIGHT
- const Fl_Grid_Align FL_GRID_CENTER = 0x0000
  
  Align the widget in the middle of the cell (default).

- const Fl_Grid_Align FL_GRID_FILL = 0x0030
  
  Stretch the widget in both directions to fill the cell.

- const Fl_Grid_Align FL_GRID_HORIZONTAL = 0x0010
  
  Stretch the widget horizontally to fill the cell.

- const Fl_Grid_Align FL_GRID_LEFT = 0x0004
  
  Align the widget at the left side of the cell.

- const Fl_Grid_Align FL_GRID_PROPORTIONAL = 0x0040
  
  Stretch the widget proportionally.

- const Fl_Grid_Align FL_GRID_RIGHT = 0x0008
  
  Align the widget at the right side of the cell.

- const Fl_Grid_Align FL_GRID_TOP = 0x0001
  
  Align the widget at the top of the cell.

- const Fl_Grid_Align FL_GRID_TOP_LEFT = FL_GRID_TOP | FL_GRID_LEFT
- const Fl_Grid_Align FL_GRID_TOP_RIGHT = FL_GRID_TOP | FL_GRID_RIGHT
- const Fl_Grid_Align FL_GRID_VERTICAL = 0x0020
  
  Stretch the widget vertically to fill the cell.

13.59.1 Detailed Description

Fl_Grid container widget.

13.60 Fl_Grid.H

Go to the documentation of this file.
00035 const Fl_Grid_Align FL_GRID_TOP = 0x0001;
00036
00038 const Fl_Grid_Align FL_GRID_BOTTOM = 0x0002;
00039
00041 const Fl_Grid_Align FL_GRID_LEFT = 0x0004;
00042
00044 const Fl_Grid_Align FL_GRID_RIGHT = 0x0008;
00045
00047 const Fl_Grid_Align FL_GRID_HORIZONTAL = 0x0010;
00048
00050 const Fl_Grid_Align FL_GRID_VERTICAL = 0x0020;
00051
00053 const Fl_Grid_Align FL_GRID_FILL = 0x0030;
00054
00056 const Fl_Grid_Align FL_GRID_PROPORTIONAL = 0x0040;
00057
00059 const Fl_Grid_Align FL_GRID_TOP_LEFT = FL_GRID_TOP | FL_GRID_LEFT;
00060 const Fl_Grid_Align FL_GRID_TOP_RIGHT = FL_GRID_TOP | FL_GRID_RIGHT;
00061 const Fl_Grid_Align FL_GRID_BOTTOM_LEFT = FL_GRID_BOTTOM | FL_GRID_LEFT;
00062 const Fl_Grid_Align FL_GRID_BOTTOM_RIGHT = FL_GRID_BOTTOM | FL_GRID_RIGHT;
00063
00147 class FL_EXPORT Fl_Grid : public Fl_Group {
00148 friend class Fl_Grid_Type;
00149
00150 public:
00151 class Cell {
00152 friend class Fl_Grid;
00153 private:
00154  Cell *next_; // next cell in row
00155  short row_; // row number
00156  short col_; // column number
00157  short rowspan_; // row span (1 - n)
00158  short colspan_; // column span (1 - n)
00159  Fl_Grid_Align align_; // widget alignment in its cell
00160  Fl_Widget *widget_; // assigned widget
00161  int w_; // minimal widget width
00162  int h_; // minimal widget height
00163
00164 public:
00165
00166 void Cell_() {
00167   next_ = NULL;
00168   row_ = 0;
00169   col_ = 0;
00170   rowspan_ = 1;
00171   colspan_ = 1;
00172   widget_ = NULL;
00173   w_ = 0;
00174   h_ = 0;
00175   align_ = 0;
00176 }
00177
00178 Cell(int row, int col) {
00179   Cell_();
00180   row_ = row;
00181   col_ = col;
00182 }
00183
00184 Cell(Fl_Widget *w, int row, int col) {
00185   Cell_();
00186   widget_ = w;
00187   row_ = row;
00188   col_ = col;
00189 }
00190
00191 ~Cell() {}  
00192
00193 Fl_Widget *widget() const { return widget_; }
00194
00195 short row() const { return row_; }
00196
00198 short col() const { return col_; }
00199
00200 void rowspan(short v) { rowspan_ = v; }
00201 void colspan(short v) { colspan_ = v; }
00202
00203 void align(Fl_Grid_Align align) { align_ = align; }
00204
00205 void minimum_size(int w, int h) {
00206   if (w>0) w_ = w; if (h>0) h_ = h;
00207   if (w > w_) if (h > h_) {
00208     // cell
00209   }
00210 private:
00211  class Row;
00212  class Col;
short rows_;
short cols_; // array of columns
short margin_left_; // left margin
short margin_top_; // top margin
short margin_right_; // right margin
short margin_bottom_; // bottom margin
short gap_row_; // gap between rows
short gap_col_; // gap between columns
Fl_Rect old_size; // only for resize callback (TBD)
Row *Rows_; // array of rows
bool need_layout_; // true if layout needs to be calculated

protected:
short gap_row_; // gap between rows
short gap_col_; // gap between columns

Fl_Color grid_color; // color for drawing the grid lines (design helper)
bool draw_grid_; // draw the grid for testing / design

protected:
void init();
Cell *add_cell(int row, int col);
void remove_cell(int row, int col);

public:
Fl_Grid(int X, int Y, int W, int H, const char *L = 0);
virtual ~Fl_Grid();

// define and manage the layout and resizing
virtual void layout(int rows, int cols, int margin = -1, int gap = -1);
virtual void layout();
virtual void clear_layout();
virtual void resize(int X, int Y, int W, int H) FL_OVERRIDE;
short rows() const { return rows_; }
short cols() const { return cols_; }

void need_layout(int set) {
    if (set) {
        need_layout_ = true;
        redraw();
    } else {
        need_layout_ = false;
    }
}
bool need_layout() const {
    return need_layout_;}

protected:
virtual void draw() FL_OVERRIDE;
void on_remove(int) FL_OVERRIDE;
virtual void draw_grid(); // draw grid lines for debugging

public:

// get and set individual margins
virtual void margin(int left, int top = -1, int right = -1, int bottom = -1);
int margin(int *left, int *top, int *right, int *bottom) const;

// get and set default row and column gaps for all rows and columns, respectively
virtual void gap(int row_gap, int col_gap = -1); // set default row and column gap(s)
void gap(int *row_gap, int *col_gap) const;

// find cells, get cell pointers
Fl_Grid::Cell * cell(int row, int col) const;
Fl_Grid::Cell * cell(Fl_Widget *widget) const;

// assign a widget to a cell
Fl_Grid::Cell * widget(Fl_Widget *wi, int row, int col, Fl_Grid_Align align = FL_GRID_FILL);
Fl_Grid::Cell * widget(Fl_Widget *wi, int row, int col, int rowspan, int colspan, Fl_Grid_Align align = FL_GRID_FILL);

// set minimal column and row sizes (widths and heights, respectively),
// set row and column specific gaps and weights
void col_width(int col, int value);
void col_width(const int *value, size_t size);
int col_width(int col) const;

void col_weight(int col, int value);
void col_weight(int col, int value);
void col_weight(const int *value, size_t size);
int col_weight(int col) const;

void col_gap(int col, int value);
void col_gap(const int *value, size_t size);
int col_gap(int col) const;

void row_height(int row, int value);
void row_height(const int *value, size_t size);
int row_height(int row) const;

void row_weight(int row, int value);
void row_weight(const int *value, size_t size);
int row_weight(int row) const;

void row_gap(int row, int value);
void row_gap(const int *value, size_t size);
int row_gap(int row) const;

int computed_col_width(int col) const;
int computed_row_height(int row) const;

void show_grid(int set) {
  draw_grid_ = set ? true : false;
}
void show_grid(int set, Fl_Color col) {
  draw_grid_ = set ? true : false;
  grid_color = col;
}

void debug(int level = 127);

}; // class Fl_Grid

#endif // _FL_FL_GRID_H_

13.61 Fl_Group.H File Reference

Fl_Group and Fl_End classes.
#include "Fl_Widget.H"

Classes

• class Fl_End

  This is a dummy class that allows you to end a Fl_Group in a constructor list of a class:

• class Fl_Group

  The Fl_Group class is the FLTK container widget.

13.61.1 Detailed Description

Fl_Group and Fl_End classes.

13.62 Fl_Group.H

Go to the documentation of this file.
#define Fl_Group_H

#include "Fl_Widget.H"

// Don't #include Fl_Rect.H because this would introduce lots of unnecessary dependencies on Fl_Rect.H

class Fl_Rect;

class FL_EXPORT Fl_Group : public Fl_Widget {

union {
  Fl_Widget** array_; // used if group has two or more children or NULL
  Fl_Widget* child1_; // used if group has one child or NULL
};

Fl_Widget* savedfocus_;
int children_; // remember initial sizes of children
int *sizes_; // remember initial sizes of children (FLTK 1.3 compat.)

int navigation(int);

static Fl_Group* current_;

// unimplemented copy ctor and assignment operator
Fl_Group(const Fl_Group&);
Fl_Group& operator=(const Fl_Group&);

protected:
void draw() FL_OVERRIDE;
void draw_child(Fl_Widget& widget) const;
void draw_children();
void draw_outside_label(const Fl_Widget& widget) const;
void update_child(Fl_Widget& widget) const;
Fl_Rect* bounds();
int *sizes(); // FLTK 1.3 compatibility
virtual int on_insert(Fl_Widget*, int);
virtual void on_remove(int);

public:

int handle(int) FL_OVERRIDE;
void begin();
void end();
static Fl_Group* current();
static void current(Fl_Group* g);
int children() const {return children_;
Fl_Widget* child(int n) const {return array()[n];
int find(const Fl_Widget* o) const {return find(&o);};
int find(const Fl_Widget& o) const {return find(&o);};
Fl_Widget** array() const;

void resize(int, int, int, int) FL_OVERRIDE;
Fl_Group(int, int, int, int, const char* = 0);
virtual ~Fl_Group();
void add(Fl_Widget&);
void add(Fl_Widget* o) {add(*o);};
void insert(Fl_Widget& o, Fl_Widget* before) {insert(o, find(before));}
void remove(int index);
void remove(Fl_Widget*);
void remove(Fl_Widget* o) {remove(*o);};
void clear();

/* delete child n (by index) */
void delete_child(int n);

virtual int delete_child(int n);

void resizable(Fl_Widget& o) {resizable_ = &o;};
void resizable(Fl_Widget* o) {resizable_ = o;};
Fl_Widget* resizable() const {return resizable_;
void add_resizable(Fl_Widget& o) {add_resizable_ = &o;};
void add_resizable(Fl_Widget* o) {add_resizable_ = o; add(o);};
void init_sizes();

void clip_children(int c) {if (c) set_flag(CLIP_CHILDREN); else clear_flag(CLIP_CHILDREN);};
unsigned int clip_children() {return flags() & CLIP_CHILDREN;};

// back compatibility functions:
virtual ~Fl_Group();
void focus(Fl_Widget* W) {W->take_focus();};

Fl_Widget* & _ddfdesign_kludge() {return resizable_;

Generated by Doxygen
// Fl_Help_Dialog.H

#ifndef Fl_Help_Dialog_H
#define Fl_Help_Dialog_H

#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Help_View.H>

class FL_EXPORT Fl_Help_Dialog {
public:
  Fl_Help_Dialog();
  ~Fl_Help_Dialog();
  int h();
  void hide();
  int load(const char *f);
  void position(int xx, int yy);
  void resize(int xx, int yy, int ww, int hh);

private:
  Fl_Double_Window *window_;
  Fl_Button *back_;
  Fl_Button *forward_;  
  Fl_Button *smaller_;  
  Fl_Button *larger_;  
  Fl_Input *find_;  
  Fl_Help_View *view_;  

  int index_;  
  int max_;  
  int line_[100];  // FIXME: we must remove those static numbers
  char file_[100][FL_PATH_MAX];  // FIXME: we must remove those static numbers
  int find_pos_;  
  int h();
  void hide();
  int load(const char *f);
  void position(int xx, int yy);
  void resize(int xx, int yy, int ww, int hh);  

  inline void cb_back__(Fl_Button *, void*);
  static void cb_back_(Fl_Button *, void*);
  inline void cb_forward__(Fl_Button *, void*);
  static void cb_forward_(Fl_Button *, void*);
  inline void cb_smaller__(Fl_Button *, void*);
  static void cb_smaller_(Fl_Button *, void*);
  inline void cb_larger__(Fl_Button *, void*);
  static void cb_larger_(Fl_Button *, void*);
  inline void cb_find__(Fl_Input *, void*);
  static void cb_find_(Fl_Input *, void*);
  inline void cb_view__(Fl_Help_View *, void*);
  static void cb_view_(Fl_Help_View *, void*);

};

#endif

// Fl_Help_Dialog dialog for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2021 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// See the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php
// generated by Fast Light User Interface Designer (fluid) version 1.0400
// DO NOT EDIT FL/Fl_Help_Dialog.H and src/Fl_Help_Dialog.cxx !!!
// ============
// and then use fluid to "write code" or edit and use fluid -c .

Generated by Doxygen
```c
void show();
void show(int argc, char **argv);
void textsize(Fl_Fontsize s);
Fl_Fontsize textsize();
void topline(const char *n);
void topline(int n);
void value(const char *f);
const char * value() const;
int visible();
int w();
int x();
int y();
};
```

13.64 Fl_Help_View.H

```
/*
file
Fl_Help_View widget . */

#ifndef Fl_Help_View_H
#define Fl_Help_View_H

#include "Fl.H"
#include "Fl_Group.H"
#include "Fl_Scrollbar.H"
#include "fl_draw.H"
#include "filename.H"

class Fl_Shared_Image;

struct Fl_Help_Block {
    const char *start, // Start of text
    +end; // End of text
    uchar border; // Draw border?
    Fl_Colorbgcolor; // Background color
    int x, // Indentation/starting X coordinate
    y, // Starting Y coordinate
    w, // Width
    h; // Height
    int line[32]; // Left starting position for each line
    int ol; // is ordered list <OL> element
    int ol_num; // item number in ordered list
};
```

```
struct Fl_Help_Link {
    char filename[192],
    name[32];
    int x,
    y,
    w,
};
```
struct FL_EXPORT Fl_Help_Font_Style {
    Fl_Font f;
    Fl_Fontsize s;
    Fl_Color c;
    void get(Fl_Font &afont, Fl_Fontsize &asize, Fl_Color &acolor) {afont=f; asize=s; acolor=c;}
    void set(Fl_Font afont, Fl_Fontsize asize, Fl_Color acolor) {f=afont; s=asize; c=acolor;}
    Fl_Help_Font_Style(Fl_Font afont, Fl_Fontsize asize, Fl_Color acolor) {set(afont, asize, acolor);}
    Fl_Help_Font_Style(){} // For in table use
};

const size_t MAX_FL_HELP_FS_ELTS = 100;

struct FL_EXPORT Fl_Help_Font_Stack {
    Fl_Help_Font_Stack() {
        nfonts_ = 0;
    }
    void init(Fl_Font f, Fl_Fontsize s, Fl_Color c) {
        nfonts_ = 0;
        elts_[nfonts_].set(f, s, c);
        fl_font(f, s);
        fl_color(c);
    }
    void top(Fl_Font &f, Fl_Fontsize &s, Fl_Color &c) { elts_[nfonts_].get(f, s, c); }
    void push(Fl_Font f, Fl_Fontsize s, Fl_Color c) {
        if (nfonts_ < MAX_FL_HELP_FS_ELTS-1) nfonts_ ++;
        elts_[nfonts_].set(f, s, c);
        fl_font(f, s); fl_color(c);
    }
    void pop(Fl_Font &f, Fl_Fontsize &s, Fl_Color &c) {
        if (nfonts_ > 0) nfonts_ --;
        top(f, s, c);
        fl_font(f, s); fl_color(c);
    }
    size_t count() const {return nfonts_;} // Gets the current number of fonts in the stack
};

protected:
    size_t nfonts_;
    Fl_Help_Font_Style elts_[MAX_FL_HELP_FS_ELTS];
};

struct Fl_Help_Target {
    char name[32];
    int y;
};

class FL_EXPORT Fl_Help_View : public Fl_Group { // Help viewer widget
    enum { RIGHT = -1, CENTER, LEFT };
void initfont(Fl_Font &f, Fl_Fontsize &s, Fl_Color &c) { f = textfont_; s = textsize_; c = textcolor_; fstack_.init(f, s, c); }
void pushfont(Fl_Font f, Fl_Fontsize s) { fstack_.push(f, s, textcolor_); }
void pushfont(Fl_Font f, Fl_Fontsize s, Fl_Color c) { fstack_.push(f, s, c); }
void popfont(Fl_Font &f, Fl_Fontsize &s, Fl_Color &c) { fstack_.pop(f, s, c); }
void add_block(const char *s, int xx, int yy, int ww, int hh, uchar border = 0);
void add_link(const char *n, int xx, int yy, int ww, int hh);
void add_target(const char *n, int yy);
static int compare_targets(const Fl_Help_Target *t0, const Fl_Help_Target *t1);
int do_align(Fl_Help_Block *block, int line, int xx, int a, int &l);
protected:
void draw() FL_OVERRIDE;
private:
void format();
void format_table(int *table_width, int *columns, const char *char *table);
void free_data();
const char *get_attr(const char *p, const char *n, char *buf, int bufsize);
Fl_Color get_color(const char *n, Fl_Color c);
Fl_Shared_Image *get_image(const char *name, int W, int H);
int get_length(const char *l);
public:
int handle(int) FL_OVERRIDE;
private:
void hv_draw(const char *t, int x, int y, int entity_extra_length = 0);
char begin_selection();
char extend_selection();
void end_selection(int c=0);
void clear_global_selection();
Fl_Help_Link *find_link(int, int);
void follow_link(Fl_Help_Link *);
Fl_Help_Block *add_block(const char *s, int xx, int yy, int ww, int hh, uchar border = 0);
Fl_Help_Block *add_block(const char *s, int xx, int yy, int ww, int hh);
void link(Fl_Help_Func *fn) { link_ = fn; }
int load(const char *f);
void resize(int, int, int, int) FL_OVERRIDE;
int scrollbar_size() const { return (scrollbar_size_); }
void scrollbar_size(int newSize) { scrollbar_size_ = newSize; }
void textcolor(Fl_Color c) { textcolor_ = c; }
Fl_Font textfont() const { return (textfont_); }
Fl_Fontsize textsize() const { return (textsize_); }
const char *title() { return (title_); }
void topline(const char *n);
void topline(int);
int topline() const { return (topline_); }
int leftline(const char *n, int extra_length = 0);
int leftline() const { return (leftline_); }
void value(const char *val);
const char *value() const { return (value_); }
void clear_selection();
void select_all();
void scrollbar_size(int newSize) { scrollbar_size_ = newSize; }
void scrollbar_size() const { return (scrollbar_size_); }
scrollbar_size_ = newSize;
}
};
#endif // !Fl_Help_View_H

Fl_Hold_Browser.H

/*
 * Fl_Hold_Browser widget . */

#ifndef Fl_Hold_Browser_H
#define Fl_Hold_Browser_H

#include "Fl_Browser.H"

class FL_EXPORT Fl_Hold_Browser : public Fl_Browser {
public:
  Fl_Hold_Browser(int X, int Y, int W, int H, const char *L=0);
};
#endif

Fl_Hor_Fill_Slider.H

/*
 * Fl_Hor_Fill_Slider widget . */

#ifndef Fl_Hor_Fill_Slider_H
#define Fl_Hor_Fill_Slider_H

#include "Fl_Slider.H"

class FL_EXPORT Fl_Hor_Fill_Slider : public Fl_Slider {
public:
  Fl_Hor_Fill_Slider(int X, int Y, int W, int H, const char *L=0);
};
#endif

Fl_Hor_Nice_Slider.H

/*
 * Fl_Hor_Nice_Slider widget . */

#ifndef Fl_Hor_Nice_Slider_H
#define Fl_Hor_Nice_Slider_H

class FL_EXPORT Fl_Hor_Nice_Slider : public Fl_Slider {
public:
  Fl_Hor_Nice_Slider(int X, int Y, int W, int H, const char *L=0);
};
#endif
Fl_Hor_Nice_Slider widget.

```cpp
class FL_EXPORT Fl_Hor_Nice_Slider : public Fl_Slider {
public:
    Fl_Hor_Nice_Slider(int X, int Y, int W, int H, const char *L=0);
};
```

Fl_Hor_Slider widget.

```cpp
class FL_EXPORT Fl_Hor_Slider : public Fl_Slider {
public:
    Fl_Hor_Slider(int X, int Y, int W, int H, const char *L=0);
};
```

Fl_Hor_Value_Slider widget.

```cpp
class FL_EXPORT Fl_Hor_Value_Slider : public Fl_Slider {
public:
    Fl_Hor_Value_Slider(int X, int Y, int W, int H, const char *L=0);
};
```


```c
#define Fl_Hor_Value_Slider_H

#include "Fl_Value_Slider.H"

class FL_EXPORT Fl_Hor_Value_Slider : public Fl_Value_Slider {
public:
  Fl_Hor_Value_Slider(int X,int Y,int W,int H,const char *l=0);
};
#endif
```

---

13.70 Fl_ICO_Image.H

---

```c
// ICO image header file for the Fast Light Tool Kit (FLTK).
// Copyright 2022-2023 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// http://www.fltk.org/COPYING.php
// Please report all bugs and problems on the following page:
// http://www.fltk.org/str.php
// The following links are provided for your convenience:
// https://en.wikipedia.org/wiki/ICO_(file_format)

#ifndef Fl_ICO_Image_H
#define Fl_ICO_Image_H
#include "Fl_BMP_Image.H"

class FL_EXPORT Fl_ICO_Image : public Fl_BMP_Image {

public:

  struct IconDirEntry {
    int bWidth;
    int bHeight;
    int bColorCount;
    int bReserved;
    int wPlanes;
    int wBitCount;
    int dwBytesInRes;
    int dwImageOffset;
  };

  Fl_ICO_Image(const char *filename, int id = -1, const unsigned char *data = NULL, const size_t
dataSize = 0);

  int idcount() const { return idcount_; }

  const IconDirEntry * icondirentry() const { return icondirentry_; }

private:

  int idcount_;  
  struct IconDirEntry *icondirentry_; 

  void load_ico_(class Fl_Image_Reader &rdr, int id);
};
#endif // Fl_ICO_Image_H
```

---

13.71 Fl_image.H File Reference

Fl_image, Fl_RGB_Image classes.
#include "Enumerations.H"
#include "Fl_Widget.H"

Classes

- class Fl_image
  
  Base class for image caching, scaling and drawing.

Generated by Oxygen
• class Fl_RGB_Image

The Fl_RGB_Image class supports caching and drawing of full-color images with 1 to 4 channels of color information.

Enumerations

• enum Fl_RGB_Scaling { FL_RGB_SCALING_NEAREST = 0, FL_RGB_SCALING_BILINEAR }

The scaling algorithm to use for RGB images.

13.71.1 Detailed Description

Fl_Image, Fl_RGB_Image classes.

13.71.2 Enumeration Type Documentation

13.71.2.1 Fl_RGB_Scaling

enum Fl_RGB_Scaling

The scaling algorithm to use for RGB images.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_RGB_SCALING_NEAREST</td>
<td>default RGB image scaling algorithm</td>
</tr>
<tr>
<td>FL_RGB_SCALING_BILINEAR</td>
<td>more accurate, but slower RGB image scaling algorithm</td>
</tr>
</tbody>
</table>

13.72 Fl_Image.H

Go to the documentation of this file.

```c
#ifndef Fl_Image_H
#define Fl_Image_H

#include "Enumerations.H"
#include "Fl_Widget.H" // for fl_uintptr_t

class Fl_Widget;
class Fl_Pixmap;
struct Fl_Menu_Item;
struct Fl_Label;
class Fl_RGB_Image;

enum Fl_RGB_Scaling {
    FL_RGB_SCALING_NEAREST = 0,
    FL_RGB_SCALING_BILINEAR
};

class Fl_EXPORT Fl_Image {

friend class Fl_Graphics_Driver;

public:
static const int ERR_NO_IMAGE = -1;
static const int ERR_FILE_ACCESS = -2;
static const int ERR_FORMAT = -3;
static const int ERR_MEMORY_ACCESS = -4;
private:
```

Generated by Doxygen
# Image Handling

The code snippet provided is from a C++ library that deals with image handling and rendering. It includes methods for creating and manipulating images, as well as functions for scaling and drawing images. Here are some key points extracted from the code:

- **Image Creation and Initialization**: The class `Fl_Image` provides constructors for creating images of specific sizes and types. It also includes methods for setting image dimensions, data, and scale sizes.

- **Image Scaling and Drawing**: The class includes methods for scaling images before drawing and for drawing the image at specific sizes and positions. These methods are critical for ensuring that images are displayed correctly across different screen sizes.

- **Image Data Handling**: The class manages image data through pointers to arrays of characters, allowing for dynamic data manipulation and access.

- **Methods and Functions**: The class includes various methods for accessing image dimensions, drawing, and releasing image resources. It also has static methods for setting and getting RGB scaling algorithms, which are used for rescaling images before drawing.

- **PlatformIndependence**: The code is designed to be platform-independent, allowing it to work across different operating systems and hardware configurations.

The `Fl_SVG_Image` class is also mentioned, indicating that the library supports SVG images, which are scalable vector graphics.

Generated by Doxygen
13.73  Fl_Image_Surface.H

```
00001 //
00002 // Draw-to-image code for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2016 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file COPYING which should have been included with this file. If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //  https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //  https://www.fltk.org/bugs.php
00015 //
00016 00017 #ifndef Fl_Image_Surface_H
00018 #define Fl_Image_Surface_H
00019 #endif
00020 #ifndef Fl_Image_Surface_H
00021 #define Fl_Image_Surface_H
00022 #endif
00023 #include <FL/Fl_Widget_Surface.H>
00024 #include <FL/Fl_Image.H>
00025 #include <FL/Fl_Shared_Image.H>
00026 #include <FL/platform_types.h> // for Fl_Offscreen
00027
00028 class FL_EXPORT Fl_Image_Surface : public Fl_Widget_Surface {
00029   friend class Fl_Graphics_Driver;
00030   private:
00031   class Fl_Image_Surface_Driver *platform_surface;
00032   Fl_Offscreen get_offscreen_before_delete();
00033   protected:
00034     void translate(int x, int y) FL_OVERRIDE;
00035     void untranslate() FL_OVERRIDE;
00036   public:
00037     Fl_Image_Surface(int w, int h, int high_res = 0, Fl_Offscreen off = 0);
00038 #ifndef Fl_Image_Surface_H
00039     Fl_Image_Surface(int w, int h, int high_res = 0, Fl_Offscreen off = 0);
00040     Fl_Image_Surface(int w, int h, int D=3, int LD=0);
00041     Fl_RGB_Image(const uchar *bits, int W, int H, int D=3, int LD=0);
00042     Fl_RGB_Image(const uchar *bits, int bits_length, int W, int H, int D, int LD);
00043     Fl_RGB_Image(const Fl_Pixmap *pxm, Fl_Color bg=FL_GRAY);
00044     ~Fl_RGB_Image() FL_OVERRIDE;
00045     Fl_RGB_Image() FL_OVERRIDE;
00046     void desaturate() FL_OVERRIDE;
00047     void draw(int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE;
00048     void draw(int X, int Y) {draw(X, Y, w(), h(), 0, 0);}
00049     void label(Fl_Widget *w) FL_OVERRIDE;
00050     void label(Fl_Menu_Item *m) FL_OVERRIDE;
00051     void uncache() FL_OVERRIDE;
00052     int cache_w() {return cache_w_;
00053     int cache_h() {return cache_h_;
00054     virtual Fl_RGB_Image *as_svg_image() { return NULL; }
00055     virtual void normalize() {}
00056   }; // Fl_Image_Surface.H
```
class FL_EXPORT Fl_Input : public Fl_Input_ {
friend class Fl_Cocoa_Screen_Driver; // Not ideal, but probably no other platform will use it

int shift_position(int p);
int shift_up_down_position(int p);
void handle_mouse(int keepmark=0);

// Private keyboard functions
int kf_lines_up(int repeat_num);
int kf_lines_down(int repeat_num);
int kf_insert_toggle();
int kf_delete_word_right();
int kf_delete_word_left();
int kf_delete_sol();
int kf_delete_eol();
int kf_delete_char_right();
int kf_delete_char_left();
int kf_move_sol();
int kf_move_eol();
int kf_clear_eol();
int kf_move_char_left();
int kf_move_char_right();
int kf_move_word_left();
int kf_move_word_right();
int kf_move_up_and_sol();
int kf_move_down_and_eol();
int kf_top();
int kf_bottom();
int kf_select_all();
int kf_undo();
int kf_redo();
int kf_copy();
int kf_paste();
int kf_copy_cut();

protected:
00257 void draw() FL_OVERRIDE;
00258 int handle_key();

};
File Documentation

13.75 Fl_Input_.H

```c
class Fl_Input_Undo_Action;
class Fl_Input_Undo_Action_List;

class FL_EXPORT Fl_Input_ : public Fl_Widget { public:
    const char * value_;        // input text
    char * buffer;              // buffer for input
    int size_;                  // size of buffer
    int bufsize;                // size of buffer
    int position_;              // position in buffer
    int mark_;                  // mark for undo
    int tab_nav_;               // tab navigation
    int xscroll_, yscroll_;      // x and y scroll
    int mu_p;                   // mouse pointer
    int maximum_size_;          // maximum size
    int shortcut_;              // shortcut
    uchar erase_cursor_only;    // erase cursor only
    Fl_Font textfont_;          // text font

private:
    Fl_Input_Undo_Action *undo_;   // undo action

    static const char *cut_menu_text;
    static const char *copy_menu_text;
    static const char *paste_menu_text;

    // FL_INPUT_DESCRIPTION
    // FL_INPUT_DESCRIPTION_WRAP
    // FL_INPUT_DESCRIPTION_READONLY
    // FL_INPUT_DESCRIPTION_WRAP_READONLY
    // FL_INPUT_DESCRIPTION_WRAP_READONLY_WRAP
    // FL_INPUT_DESCRIPTION_WRAP_READONLY_WRAP_WRAP
    // FL_INPUT_DESCRIPTION_WRAP_READONLY_WRAP_WRAP_WRAP

public:
    int handle_rmb();

    int handle(int) FL_OVERRIDE;
    Fl_Input(int,int,int,int,const char * = 0);

    static const char *cut_menu_text;
    static const char *copy_menu_text;
    static const char *paste_menu_text;

    #endif

#define FL_NORMAL_INPUT 0
#define FL_FLOAT_INPUT 1
#define FL_INT_INPUT 2
#define FL_HIDDEN_INPUT 3
#define FL_MULTILINE_INPUT 4
#define FL_SECRET_INPUT 5
#define FL_INPUT_TYPE 7
#define FL_INPUT_READONLY 8
#define FL_NORMAL_OUTPUT (FL_NORMAL_INPUT | FL_INPUT_READONLY)
#define FL_MULTILINE_OUTPUT (FL_MULTILINE_INPUT | FL_INPUT_READONLY)
#define FL_MULTILINE_INPUT_WRAP (FL_MULTILINE_INPUT | FL_INPUT_WRAP)
#define FL_MULTILINE_OUTPUT_WRAP (FL_MULTILINE_INPUT | FL_INPUT_READONLY | FL_INPUT_WRAP)
```

Generated by Doxygen
Fl_Fontsize
Fl_Color
Fl_Color
Fl_Input_Undo_Action *
Fl_Input_Undo_Action_List *
Fl_Input_Undo_Action_List *
static double up_down_pos;
static int was_up_down;

/* Convert a given text segment into the text that will be rendered on screen. */
const char * expand(const char*, char*) const;

/* Calculates the width in pixels of part of a text buffer. */
double expandpos(const char*, const char*, const char*, int*) const;

/* Mark a range of characters for update. */
void minimal_update(int, int);

/* Mark a range of characters for update. */
void minimal_update(int p);

/* Copy the value from a possibly static entry into the internal buffer. */
void put_in_buffer(int newsize);

/* Set the current font and font size. */
void setfont() const;

protected:

/* Find the start of a word. */
int word_start(int i) const;

/* Find the end of a word. */
int word_end(int i) const;

/* Find the start of a line. */
int line_start(int i) const;

/* Find the end of a line. */
int line_end(int i) const;

/* Draw the text in the passed bounding box. */
void drawtext(int, int, int, int);

/* Draw the text in the passed bounding box. */
void drawtext(int, int, int, int, bool draw_active);

/* Move the cursor to the column given by up_down_pos. */
int up_down_position(int, int keepmark=0);

/* Handle mouse clicks and mouse moves. */
void handlemouse(int, int, int, int, int keepmark=0);

/* Handle all kinds of text field related events. */
int handletext(int e, int, int, int, int);

/* Check the when() field and do a callback if indicated. */
void maybe_do_callback(Fl_Callback_Reason reason = FL_REASON_UNKNOWN);

int xscroll() const {return xscroll_;} [return xscroll_;]
int yscroll() const {return yscroll_;}
void yscroll(int yOffset) { yscroll_ = yOffset; damage(FL_DAMAGE_EXPOSE);}

/* Return the number of lines displayed on a single page. */
int linesPerPage();

/* Apply the current undo/redo operation, called from undo() or redo() */
int apply_undo();

public:

/* Change the size of the widget. */
void resize(int, int, int) FL_OVERRIDE;

/* Constructor */
Fl_Input_(int, int, int, int, const char * = 0);

/* Destructor */
~Fl_Input_();

/* Changes the widget text. */
int value(const char *); // Changes the widget text.
int value(const char *, int); // Changes the widget text.
int value(int value); // Changes the widget text.
int value(const char *, int); // Changes the widget text.
int value(int value); // Changes the widget text.
int value(double value); // Changes the widget text.
int static_value(const char *); // Changes the widget text.
int static_value(const char *, int); // Changes the widget text.
const char * value() const {return value_;} // Changes the widget text.
int ivalue() const; // Changes the widget text.
double dvalue() const; // Changes the widget text.

/* Returns the Unicode character at index \p i. */
unsigned int index(int i) const;

int size() const {return size_;} // Changes the widget text.
void size(int W, int H) { Fl_Widget::size(W, H); } // Changes the widget text.
int maximum_size() const {return maximum_size_;} // Changes the widget text.
void maximum_size(int m) {maximum_size_ = m;} // Changes the widget text.
int insert_position() const { return position_; } // Changes the widget text.
int position() const { return insert_position(); } // Changes the widget text.
int mark() const {return mark_;} // Changes the widget text.

/* Sets the index for the cursor and mark. */
int insert_position(int p, int m);

FL_DEPRECATED("in 1.4.0 - use insert_position() instead",
int position() const { return insert_position(); } // Changes the widget text.
int insert_position(int p, int m); // Changes the widget text.
FL_DEPRECATED("in 1.4.0 - use insert_position(p, m) or Fl_Widget::position(x, y) instead",
int position(int p, int m) { return insert_position(p, m); } // Changes the widget text.
int insert_position(int p) { return insert_position(p, p); } // Changes the widget text.
FL_DEPRECATED("in 1.4.0 - use insert_position(p) instead",
int position(int p) { return insert_position(p); } // Changes the widget text.

int insert(const char * t, int l=0){return replace(position_, mark_, t, l);} // Changes the widget text.
int append(const char * t, int l=0, char keep_selection=0); // Changes the widget text.
int copy(int clipboard); // Changes the widget text.
int undo(); // Changes the widget text.
bool can_undo() const; // Changes the widget text.
int redo(); // Changes the widget text.
bool can_redo() const; // Changes the widget text.
int copy_cuts(); // Changes the widget text.
int shortcut() const {return shortcut_;} // Changes the widget text.
void shortcut(int s) {shortcut_ = s;} // Changes the widget text.
13.76 Fl_Input_Choice.H

```cpp
Fl_Font textfont() const { return textfont_; }
void textfont(Fl_Font s) { textfont_ = s; }
Fl_Fontsize textsize() const { return textsize_; }
void textsize(Fl_Fontsize s) { textsize_ = s; }
Fl_Color textcolor() const { return textcolor_; }
void textcolor(Fl_Color n) { textcolor_ = n; }
Fl_Color cursor_color() const { return cursor_color_; }
void cursor_color(Fl_Color n) { cursor_color_ = n; }
int input_type() const { return type() & FL_INPUT_TYPE; }
void input_type(int t) { type((uchar)(t | readonly())); }
int readonly() const { return type() & FL_INPUT_READONLY; }
void readonly(int b) { if (b) type((uchar)(type() | FL_INPUT_READONLY));
  else type((uchar)(type() & ~FL_INPUT_READONLY)); }
int wrap() const { return type() & FL_INPUT_WRAP; }
void wrap(int b) { if (b) type((uchar)(type() | FL_INPUT_WRAP));
  else type((uchar)(type() & ~FL_INPUT_WRAP)); }
void tab_nav(int val) {
  tab_nav_ = val;
}
int tab_nav() const {
  return tab_nav_; }
};
#endif
```

13.76 Fl_Input_Choice.H

```
// Fl_Input_Choice widget.
#ifndef Fl_Input_Choice_H
#define Fl_Input_Choice_H
#include <FL/Fl.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Menu_Button.H>

// A combination of the input widget and a menu button.
// The user can either type into the input area, or use the menu button chooser on the right to choose an item which loads the input area with the selected text.
// Note: doxygen docs in src/Fl_Input_Choice.cxx
```

Generated by Doxygen
class FL_EXPORT Fl_Input_Choice : public Fl_Group {

    // Private class to handle slightly 'special' behavior of menu button
    class InputMenuButton : public Fl_Menu_Button {
        void draw() FL_OVERRIDE;
        const Fl_Menu_Item * popup();
    public:
        InputMenuButton(int X, int Y, int W, int H, const char *L=0);
        int handle(int e) FL_OVERRIDE;
    };

    Fl_Input *inp_;
    InputMenuButton *menu_;

    // note: this is used by the Fl_Input_Choice ctor.
    static void menu_cb(Fl_Widget *, void *data);

    // note: this is used by the Fl_Input_Choice ctor.
    static void inp_cb(Fl_Widget *, void *data);

    protected:
        // Custom resize behavior -- input stretches, menu button doesn't.
        virtual int inp_x() const { return(x() + Fl::box_dx(box())); }
        virtual int inp_y() const { return(y() + Fl::box_dy(box())); }
        virtual int inp_w() const { return(w() - Fl::box_dw(box()) - 20); }
        virtual int inp_h() const { return(h() - Fl::box_dh(box())); }
        virtual int menu_x() const { return(x() + w() - 20 - Fl::box_dx(box())); }
        virtual int menu_y() const { return(y() + Fl::box_dy(box())); }
        virtual int menu_w() const { return(20); }
        virtual int menu_h() const { return(h() - Fl::box_dh(box())); }

    public:
        Fl_Input_Choice(int X, int Y, int W, int H, const char *L=0);
        void resize(int X, int Y, int W, int H) FL_OVERRIDE;
        void add(const char *s) { menu_->add(s); }
        int changed() const { return inp_->changed() | Fl_Widget::changed(); }
        void clear_changed();
        void set_changed();
        void clear() { menu_->clear(); }
        Fl_Boxtype down_box() const { return (menu_->down_box()); }
        void down_box(Fl_Boxtype b) { menu_->down_box(b); }
        const Fl_Menu_Item *menu() { return (menu_->menu()); }
        void menu(const Fl_Menu_Item *m) { menu_->menu(m); }
        Fl_Color textcolor() const { return (inp_->textcolor()); }
        void textcolor(Fl_Color c) { inp_->textcolor(c); }
        Fl_Font textfont() const { return (inp_->textfont()); }
        void textfont(Fl_Font f) { inp_->textfont(f); }
        Fl_Fontsize textsize() const { return (inp_->textsize()); }
        void textsize(Fl_Fontsize s) { inp_->textsize(s); }
        const char * value() const { return (inp_->value()); }
        void value(const char *val) { inp_->value(val); }
        Fl_Menu_Button *menubutton() { return menu_; }
        Fl_Input *input() { return inp_; }

    protected:
        // Custom resize behavior -- input stretches, menu button doesn't.
        void add(const char *s) { menu_->add(s); }
        int changed() const { return inp_->changed() | Fl_Widget::changed(); }
        void clear_changed();
        void set_changed();
        void clear() { menu_->clear(); }
        Fl_Boxtype down_box() const { return (menu_->down_box()); }
        void down_box(Fl_Boxtype b) { menu_->down_box(b); }
        const Fl_Menu_Item *menu() { return (menu_->menu()); }
        void menu(const Fl_Menu_Item *m) { menu_->menu(m); }
        Fl_Color textcolor() const { return (inp_->textcolor()); }
        void textcolor(Fl_Color c) { inp_->textcolor(c); }
        Fl_Font textfont() const { return (inp_->textfont()); }
        void textfont(Fl_Font f) { inp_->textfont(f); }
        Fl_Fontsize textsize() const { return (inp_->textsize()); }
        void textsize(Fl_Fontsize s) { inp_->textsize(s); }
        const char * value() const { return (inp_->value()); }
        void value(const char *val) { inp_->value(val); }
        Fl_Menu_Button *menubutton() { return menu_; }
        Fl_Input *input() { return inp_; }

    };
13.77 Fl_Int_Input.H

```c

#include "Fl_Input.H"

class FL_EXPORT Fl_Int_Input : public Fl_Input {
public:
    Fl_Int_Input(int X, int Y, int W, int H, const char *l = 0);
};
```

13.78 Fl_JPEG_Image.H

```c

#include "Fl_RGB_Image.H"

class FL_EXPORT Fl_JPEG_Image : public Fl_RGB_Image {
public:
    Fl_JPEG_Image(const char *filename);
    Fl_JPEG_Image(const char *name, const unsigned char *data, int data_length=-1);

protected:
    void load_jpg_(const char *filename, const char *sharename, const unsigned char *data, int data_length=-1);
};
```

13.79 Fl_Light_Button.H

```c

#include "...
```

Generated by Doxygen
class FL_EXPORT Fl_Light_Button : public Fl_Button {
protected:
  void draw() FL_OVERRIDE;
public:
  int handle(int) FL_OVERRIDE;
  Fl_Light_Button(int x,int y,int w,int h,const char *l = 0);
};

#endif

---

13.80 Fl_Line_Dial.H

---

13.81 Fl_Menu.H

---
Fl_Menu_.H

Menu base class header file for the Fast Light Tool Kit (FLTK).

Copyright 1998-2019 by Bill Spitzak and others.

This library is free software. Distribution and use rights are outlined in
the file "COPYING" which should have been included with this file. If this
file is missing or damaged, see the license at:
https://www.fltk.org/COPYING.php

Please see the following page on how to report bugs and issues:
https://www.fltk.org/bugs.php

file

ifndef Fl_Menu__H
#define Fl_Menu__H

ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif
#include "Fl_Menu_Item.H"

class FL_EXPORT Fl_Menu_ : public Fl_Widget {

Fl_Menu_Item *menu_
const Fl_Menu_Item *value_
const Fl_Menu_Item *prev_value_

protected:
uchar alloc; // flag indicates if menu_ is a dynamic copy (=1) or not (=0)
uchar down_box_
Fl_Boxtype menu_box_
Fl_Font textfont_
Fl_Fontsize textsize_
Fl_Color textcolor_

int item_pathname_(char *name, int namelen, const Fl_Menu_Item *finditem, const Fl_Menu_Item *menu=0) const;

public:
Fl_Menu_(int,int,int,int,const char *__);
~Fl_Menu_();
int item_pathname(char *name, int namelen, const Fl_Menu_Item *finditem=0) const;
const Fl_Menu_Item * picked(const Fl_Menu_Item*);
const Fl_Menu_Item * find_item(const char *name);
const Fl_Menu_Item * find_item(Fl_Callback*);
const Fl_Menu_Item * find_item_with_user_data(void*);
const Fl_Menu_Item * find_item_with_argument(long);
int find_index(const char *name) const;
int find_index(const Fl_Menu_Item *item) const;
int find_index(Fl_Callback *cb) const;

const Fl_Menu_Item * test_shortcut() {return picked(menu()->test_shortcut());}
void global();

const Fl_Menu_Item *menu() const {return menu_}
const Fl_Menu_Item *menu_end(); // in src/Fl_Menu_add.cxx
void menu(const Fl_Menu_Item *m);
void copy(const Fl_Menu_Item *m, void* user_data = 0);
int insert(int index, const char *, int shortcut, Fl_Callback*, void* = 0, int = 0);
in src/Fl_Menu_add.cxx
int insert(int index, const char * a, const char* b, Fl_Callback* c, void* d = 0, int e = 0) {
int i = insert(index, a, fl_old_shortcut(b), c, d, e);
return i;
}
int add(const char * a, const char * b, Fl_Callback * c, void* d = 0, int e = 0) { // see src/Fl_Menu_add.cxx
int i = add(a, fl_old_shortcut(b), c, d, e);
return i;
}

const Fl_Menu_Item* mvalue() const {return value_;
void clear_submenu(int index);
void clear();
int clear_submenu(int index);
void insert(int index, const char * a, const char * b, Fl_Callback * c, void* d = 0, int e = 0) {
return insert(index, a, fl_old_shortcut(b), c, d, e);
}
int add(const char *);
int size() const {
size(int W, int H) {
int i = add(const char * a, const char * b, Fl_Callback * c, void* d = 0, int e = 0) {
return insert(index, a, fl_old_shortcut(b), c, d, e);
}
int add(const char *);
int size() const {
size(int W, int H) {
int i = add(const char * a, const char * b, Fl_Callback * c, void* d = 0, int e = 0) {
return insert(index, a, fl_old_shortcut(b), c, d, e);
}
int add(const char *};
int size() const {
size(int W, int H) {
int i = add(const char * a, const char * b, Fl_Callback * c, void* d = 0, int e = 0) {
return insert(index, a, fl_old_shortcut(b), c, d, e);
}
int add(const char *};
const Fl_Menu_Item *prev_mvalue() const { return prev_value_; }
int value() const { return value_ ? (int)(value_-menu_) : -1; }
int value(const Fl_Menu_Item *);
int value(int i) { return value(menu_+i); }
const char *text() const { return value_ ? value_->text : 0; }
const char *text(int i) const { return menu_[i].text; }
Fl_Font textfont() const { return textfont_; }
void textfont(Fl_Font c) { textfont_ = c; }
Fl_Fontsize textsize() const { return textsize_; }
void textsize(Fl_Fontsize c) { textsize_ = c; }
Fl_Color textcolor() const { return textcolor_; }
void textcolor(Fl_Color c) { textcolor_ = c; }
Fl_Boxtype down_box() const { return (Fl_Boxtype)down_box_; }
void down_box(Fl_Boxtype b) { down_box_ = b; }
Fl_Boxtype menu_box() const { return menu_box_; }
void menu_box(Fl_Boxtype b) { menu_box_ = b; }
Fl_Color down_color() const { return selection_color(); }
void down_color(unsigned c) { selection_color(c); }
void setonly(Fl_Menu_Item * item);  }

13.83  Fl_Menu_Bar.H

00016 /* Fl_Menu_Bar widget. */
00017 Fl_Menu_Bar widget . */
00019 Fl_Menu_Bar widget . */
00020 ifndef Fl_Menu_Bar_H
00021 #define Fl_Menu_Bar_H
00022 #ifndef Fl_Menu_Bar_H
00023 #include "Fl_Menu_.H"
00024 #define Fl_Menu_Bar_H
00025 #endif
00026 Fl_EXPCLASS Fl_Menu_Bar : public Fl_Menu_ {
00027 friend class Fl_Sys_Menu_Bar_Driver;
00028 protected:
00029 void draw() FL_OVERRIDE;
00030 public:
00031 int handle(int) FL_OVERRIDE;
00032 Fl_Menu_Bar(int X, int Y, int W, int H, const char *l=0);  
00033 virtual void update() {} 
00034 };
00035 #endif

13.84  Fl_Menu_Button.H

00036 Fl_EXPCLASS Fl_Menu_Bar : public Fl_Menu_ {
00037 friend class Fl_Sys_Menu_Bar_Driver;
00038 protected:
00039 void draw() FL_OVERRIDE;
00040 public:
00041 int handle(int) FL_OVERRIDE;
00042 Fl_Menu_Bar(int X, int Y, int W, int H, const char *l=0);  
00043 virtual void update() {} 
00044 };
00045 #endif
```c
#include <FL/Fl_Widget.H>
#include <FL/Fl_Image.H>
#include <FL/Fl_Multi_Label.H>
#include <FL/platform_types.h>

Classes

• struct Fl_Menu_Item

  The Fl_Menu_Item structure defines a single menu item that is used by the Fl_Menu_ class.

Typedefs

• typedef Fl_Menu_Item Fl_Menu

Enumerations

• enum {
  FL_MENU_INACTIVE = 1, FL_MENU_TOGGLE = 2, FL_MENU_VALUE = 4, FL_MENU_RADIO = 8,
  FL_MENU_INVISIBLE = 0x10, FL_SUBMENU_POINTER = 0x20, FL_SUBMENU = 0x40, FL_MENU_DIVIDER
  = 0x80,
  FL_MENU_HORIZONTAL = 0x100, FL_MENU_RESERVED = 0xffffffff
}

• enum {
  FL_PUP_NONE = 0, FL_PUP_GREY = FL_MENU_INACTIVE, FL_PUP_GRAY = FL_MENU_INACTIVE,
  FL_MENU_BOX = FL_MENU_TOGGLE,
  FL_PUP_BOX = FL_MENU_TOGGLE, FL_MENU_CHECK = FL_MENU_VALUE, FL_PUP_CHECK = FL_PUP
  _VALUE, FL_PUPADIO = FL_MENU_RADIO, FL_PUP_INVISIBLE = FL_MENU_INVISIBLE, FL_PUP_SUBMENU
  = FL_SUBMENU_POINTER
}

Functions

• Fl_Shortcut fl_old_shortcut (const char *)

  Emulation of XForms named shortcuts.
```
## 13.85.1 Enumeration Type Documentation

### 13.85.1.1 anonymous enum

**anonymous enum**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_MENU_INACTIVE</td>
<td>Deactivate menu item (gray out)</td>
</tr>
<tr>
<td>FL_MENU_TOGGLE</td>
<td>Item is a checkbox toggle (shows checkbox for on/off state)</td>
</tr>
<tr>
<td>FL_MENU_VALUE</td>
<td>The on/off state for checkbox/radio buttons (if set, state is 'on')</td>
</tr>
<tr>
<td>FL_MENU_RADIO</td>
<td>Item is a radio button (one checkbox of many can be on)</td>
</tr>
<tr>
<td>FL_MENU_INVISIBLE</td>
<td>Item will not show up (shortcut will work)</td>
</tr>
<tr>
<td>FL_SUBMENU_POINTER</td>
<td>Indicates user_data() is a pointer to another menu array.</td>
</tr>
<tr>
<td>FL_SUBMENU</td>
<td>Item is a submenu to other items.</td>
</tr>
<tr>
<td>FL_MENU_DIVIDER</td>
<td>Creates divider line below this item. Also ends a group of radio buttons.</td>
</tr>
<tr>
<td>FL_MENU_HORIZONTAL</td>
<td>??? – reserved, internal (do not use)</td>
</tr>
<tr>
<td>FL_MENU_RESERVED</td>
<td>These bits are reserved for internal or future usage (do not use)</td>
</tr>
</tbody>
</table>

### 13.86 Fl_Menu_Item.H

Go to the documentation of this file.

```c
#ifndef Fl_Menu_Item_H
#define Fl_Menu_Item_H

#include <FL/Fl_Widget.H>
#include <FL/Fl_Image.H>
#include <FL/Fl_Multi_Label.H>
#include <FL/platform_types.h> // for FL_COMMAND and FL_CONTROL

// doxygen needs the following line to enable e.g. ::FLMENU_TOGGLE to link to the enums

enum { // values for flags:
  FL_MENU_INACTIVE = 1,
  FL_MENU_TOGGLE = 2,
  FL_MENU_VALUE = 4,
  FL_MENU_RADIO = 8,
  FL_MENU_INVISIBLE = 0x10,
  FL_SUBMENU_POINTER = 0x20,
  FL_SUBMENU = 0x40,
  FL_MENU_DIVIDER = 0x80,
  FL_MENU_HORIZONTAL = 0x100,
  FL_MENU_RESERVED = 0xffffffff
};

extern FL_EXPORT Fl_Shortcut fl_old_shortcut(const char*);

class Fl_Menu_
{
  Fl_Menu_
  struct FL_EXPORT Fl_Menu_Item {
    const char *text;
    int shortcut_; // reserved, internal (do not use)
    int flags;
    int labeltype_;
  };
};
```

Generated by Doxygen
Fl_Font labelfont_;  
Fl_Fontsize labelsize_;  
Fl_Color labelcolor_;  

// advance N items, skipping submenus:  
const Fl_Menu_Item *next(int i=1) const;  

const Fl_Menu_Item *next() const { return next(1); }  

const Fl_Menu_Item *first() const { return next(0); }  

// methods on menu items:  

const char * label() const { return text; }  

void label(const char * a) { text = a; }  

void label(Fl_Labeltype a, const char * b) { labeltype_ = a; text = b; }  

void multi_label(const Fl_Multi_Label *ml) { label(FL_MULTI_LABEL, (const char *)ml); }  

void image_label(const Fl_Image *image) { label(FL_IMAGE_LABEL, (const char *)image); }  

Fl_Labeltype labeltype() const {return (Fl_Labeltype)labeltype_;}  

void labeltype(Fl_Labeltype a) {labeltype_ = a;}  

Fl_Color labelcolor() const {return labelcolor_;}  

void labelcolor(Fl_Color a) {labelcolor_ = a;}  

Fl_Font labelfont() const {return labelfont_;}  

void labelfont(Fl_Font a) {labelfont_ = a;}  

Fl_Fontsize labelsize() const {return labelsize_;}  

void labelsize(Fl_Fontsize a) {labelsize_ = a;}  

Fl_Callback_p callback() const {return callback_;}  

void callback(Fl_Callback * c, void* p) {callback_=(Fl_Callback*)_c; user_data_=(void*)(fl_intptr_t)p;}  

void * user_data() const {return user_data_;}  

long argument() const {return (long)(fl_intptr_t)user_data_;}  

void argument(long v) {user_data_ = (void *)(fl_intptr_t)v;}  

int shortcut() const {return shortcut_;}  

int submenu() const {return (flags&FL_SUBMENU|FL_SUBMENU_POINTER);}  

int checkbox() const {return flags&FL_MENU_TOGGLE;}  

int radio() const {return flags&FL_MENU_RADIO;}  

int value() const {return (flags & FL_MENU_VALUE) ? 1 : 0;}  

int active() const {return !(flags&FL_MENU_INACTIVE);}  

int visible() const {return !(flags&FL_MENU_INVISIBLE);}  

void show();  

void hide();  

int active();  

int setonly(Fl_Menu_Item const * first = NULL);  

int visible() const {return !flags&FL_MENU_INVISIBLE;}  

void show() {flags |= ~FL_MENU_INVISIBLE;}  

void hide() {flags |= FL_MENU_INVISIBLE;}  

int active() const {return !flags&FL_MENU_INACTIVE;}
00430 void activate() {flags &= ~FL_MENU_INACTIVE;}
00435 void deactivate() {flags |= FL_MENU_INACTIVE;}
00437 int activevisible() const {return !(flags & (FL_MENU_INACTIVE|FL_MENU_INVISIBLE));}
00438 // compatibility for FLUID so it can set the image of a menu item...
00440 void image(Fl_Image* image) {image->label(this);}
00445 void image(Fl_Image& image) {image.label(this);}
00446 // used by menubar:
00453 // popup menus without using an Fl_Menu_ widget:
00456 const Fl_Menu_Item* popup(
00457 int x, int y,
00459 int measure(int* h, const Fl_Menu_*) const;
00461 int X, int Y,
00465 void draw(int x, int y, int w, int h, const Fl_Menu_* int t=0) const;
00468 const Fl_Menu_Item* title = 0,
00470 Fl_Menu_Item* testShortcut() const;
00472 Fl_Menu_Item* findShortcut(int* ip=0, const bool require_alt = false) const;
00473 // popup menus without using an Fl_Menu_ widget:
00474 const Fl_Menu_Item* popup(
00477 #endif
00503 int checked() const {return value();}
00511 inline void check(const set();}
00520 inline void uncheck(const clear();}
00521 int insert(int const char*, int, Fl_Callback*, void* = 0, int = 0);
00522 int add(const char*, int shortcut, Fl_Callback*, void* = 0, int = 0);
00525 int size() const;
00529 inline int checked() const {return value();}
00531 inline void check() {set();}
00535 enum { // back-compatibility enum:
00539 int add(const char a, const char b, Fl_Callback* c,
00549 typedef Fl_Menu_Item Fl_Menu; // back compatibility
00553 enum { // back-compatibility enum:
00556 int add(const char* a, int, Fl_Callback*, void* = 0, int = 0);
00557 void doCallback(Fl_Widget* o, const char* title = 0,
00567 int measure(int*, const Fl_Menu_*) const;
00570 int measure(int*, const Fl_Menu_*) const;
00576 int add(const char a, const char b, Fl_Callback* c,
00581 inline int checked() const {return value();}
00582 inline void check() {set();}
00583 inline void uncheck() {clear();}
00587 void doCallback(Fl_Widget* o, void* arg) const {Fl::callback_reason_ = FL_REASON_SELECTED;
00594 #ifndef _FLMENU_H
13.87 Fl_Menu_Window.H
00001 // Menu window header file for the Fast Light Tool Kit (FLTK).
00003 // Copyright 1998-2010 by Bill Spitzak and others.
00005 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file. If this
00008 // library is damaged, see the license at:
00009 // https://www.fltk.org/COPYING.php
00011 // Please see the following page on how to report bugs and issues:
00012 // Generated by Doxygen

13.87 Fl_Menu_Window.H
0001 // Menu window header file for the Fast Light Tool Kit (FLTK).
0003 // Copyright 1998-2010 by Bill Spitzak and others.
0005 // This library is free software. Distribution and use rights are outlined in
0007 // the file "COPYING" which should have been included with this file. If this
0008 // library is damaged, see the license at:
0009 // https://www.fltk.org/COPYING.php
0011 // Please see the following page on how to report bugs and issues:
13.88 fl_message.H

```c
#ifndef Fl_Menu_Window_H
#define Fl_Menu_Window_H

#include "Fl_Single_Window.H"

class FL_EXPORT Fl_Menu_Window : public Fl_Single_Window {
public:
  ~Fl_Menu_Window();
  Fl_Menu_Window(int W, int H, const char *l = 0);
  Fl_Menu_Window(int X, int Y, int W, int H, const char *l = 0);
};
#endif
```

13.88 fl_message.H

```c
#ifndef _FL_fl_message_H_
#define _FL_fl_message_H_

#include "fl_ask.H"

#endif
```

13.89 Fl_Multi_Browser.H

```c
#ifndef Fl_Multi_Browser_H
#define Fl_Multi_Browser_H

#include "Fl_Browser.H"

class FL_EXPORT Fl_Multi_Browser : public Fl_Browser {
public:
  Fl_Multi_Browser(int X,int Y,int W,int H,const char *L=0);
};
#endif
```

13.89 Fl_Multi_Browser.H
13.90  Fl_Multi_Label.H

```c
/*
 * Fl_Multi_Label widget.
 */

#ifndef Fl_Multi_Label_H
#define Fl_Multi_Label_H

class Fl_Widget;
struct Fl_Menu_Item;

struct FL_EXPORT Fl_Multi_Label {
    const char * labela;
    const char * labelb;
    uchar typea;
    uchar typeb;

    void label(Fl_Widget *);
    void label(Fl_Menu_Item *);
};
#endif
```

13.91  Fl_Multiline_Input.H

```c
#include "Fl_Input.H"

class FL_EXPORT Fl_Multiline_Input : public Fl_Input {
public:
    Fl_Multiline_Input(int X, int Y, int W, int H, const char *l = 0);
};
```

13.92  Fl_Multiline_Output.H

```c
#include "Fl_Input.H"

class FL_EXPORT Fl_Multiline_Output : public Fl_Input {
public:
    Fl_Multiline_Output(int X, int Y, int W, int H, const char *l = 0);
};
```
13.93 Fl_Native_File_Chooser.H File Reference

Fl_Native_File_Chooser widget.

```
#include <FL/Fl_Export.H>
#include <FL/Fl_File_Chooser.H>
```

Classes

- class Fl_Native_File_Chooser

  This class lets an FLTK application easily and consistently access the operating system's native file chooser.

13.93.1 Detailed Description

Fl_Native_File_Chooser widget.

13.94 Fl_Native_File_Chooser.H

Go to the documentation of this file.
and the KDE file chooser runs under the KDE desktop.

class Fl_Quartz_Native_File_Chooser_Driver  
<== this API implementation runs a Mac OS X file chooser

class Fl_WinAPI_Native_File_Chooser_Driver  
<== this API implementation runs a Windows file chooser

each platform must implement the constructor of the Fl_Native_File_Chooser class.

This particular implementation:

Fl_Native_File_Chooser::Fl_Native_File_Chooser(int val) {
  platform_fnfc = new Fl_Native_File_Chooser_FLTK_Driver(val);
}

This other implementation:

Fl_Native_File_Chooser::Fl_Native_File_Chooser(int val) {
  platform_fnfc = 0;
}

No more code is required. The cross-platform Fl_Native_File_Chooser_FLTK.cxx file must be compiled in libfltk,
and the default FLTK file chooser will be used.

This other implementation:

Fl_Native_File_Chooser::Fl_Native_File_Chooser(int val) {
  platform_fnfc = 0;
}

can be used by any platform.

No more code is required. The cross-platform Fl_Native_File_Chooser_FLTK.cxx file must be compiled in libfltk,
and the default FLTK file chooser will be used.

This other implementation:

Fl_Native_File_Chooser::Fl_Native_File_Chooser(int val) {
  platform_fnfc = 0;
}

can be used by a platform that needs no file chooser.

**/

ifndef FL_NATIVE_FILE_CHOOSER_H
#define FL_NATIVE_FILE_CHOOSER_H

#include <FL/Fl_Export.H>
#include <FL/Fl_File_Chooser.H>

class Fl_Native_File_Chooser_Driver;

class FL_EXPORT Fl_Native_File_Chooser {
private:
  enum Type {
    BROWSE_FILE = 0,
    BROWSE_DIRECTORY,
    BROWSE_MULTI_FILE,
    BROWSE_MULTI_DIRECTORY,
    BROWSE_SAVE_FILE,
    BROWSE_SAVE_DIRECTORY
  };
  enum Option {
    NO_OPTIONS = 0x0000,
    SAVEAS_CONFIRM = 0x0001,
    NEW_FOLDER = 0x0002,
    PREVIEW = 0x0004,
    USE_FILTER_EXT = 0x0008
  };
  static const char *file_exists_message;

public:
  Fl_Native_File_Chooser(int val = BROWSE_FILE); // each platform implements it
  ~Fl_Native_File_Chooser();
  void type(int t);
  int type() const;
  void options(int o);
  int options() const;
  int count() const;
  const char *filename() const;
  const char *filename(int i) const;
  void directory(const char *val) ;
  const char *directory() const;
  void title(const char *t);
  const char * title() const;
  const char *filter() const ;
  void filter(const char *f);
  int filters() const ;
  void filter_value(int i) ;
  int filter_value() const ;
  void preset_file(const char *f);
  const char * preset_file() const;
  const char *errmsg() const ;
  int show() ;
};

class Fl_Native_File_Chooser_Driver {
private:
  static void chrcat(char *s, char c);
  static char *strapp(char *s, const char *val);
  static char *strfree(char *val);
  static char *strnew(const char *val);
};

Generated by Doxygen
public:
    FL_Native_File_Chooser_Driver(int) {}
    virtual ~FL_Native_File_Chooser_Driver() {}
    virtual void type(int) {}
    virtual int type() const {return 0;}
    virtual void options(int) {}
    virtual int options() const {return 0;}
    virtual int count() const {return 0;}
    virtual const char *filename() const {return 0;}
    virtual const char *filename(int) const {return 0;}
    virtual void directory(const char *) {}
    virtual const char *directory() const {return 0;}
    virtual void title(const char *) {}
    virtual const char *title() const {return 0;}
    virtual const char *filter() const {return 0;}
    virtual void filter(const char *) {}
    virtual int filters() const {return 0;}
    virtual void filter_value(int) {}
    virtual int filter_value() const {return 0;}
    virtual void preset_file(const char *) {}
    virtual const char *preset_file() const {return 0;}
    virtual const char *errmsg() const {return 0;}
    virtual int show() {return 1;}
};

class FL_Native_File_Chooser_FLTK_Driver : public FL_Native_File_Chooser_Driver {
    private:
    void errmsg(const char *msg);
    int type_fl_file(int val);
    int exist_dialog();
    void parse_filter();
    protected:
    int _btype; // kind-of browser to show()
    int _options; // general options
    int _filters;
    char *_filter; // user supplied filter
    char *_parsedfilt; // parsed filter
    int _filtvalue; // selected filter
    char *_preset_file;
    char *_prevvalue; // Returned filename
    char *_directory;
    char *_errmsg; // error message
    Fl_File_Chooser *_file_chooser;
    public:
    FL_Native_File_Chooser_FLTK_Driver(int val);
    virtual ~FL_Native_File_Chooser_FLTK_Driver();
    virtual void type(int t) FL_OVERRIDE;
    virtual int type() const FL_OVERRIDE;
    virtual void options(int o) FL_OVERRIDE;
    virtual int options() const FL_OVERRIDE;
    virtual int count() const FL_OVERRIDE;
    virtual const char *filename() const FL_OVERRIDE;
    virtual const char *filename(int i) const FL_OVERRIDE;
    virtual void directory(const char *val) FL_OVERRIDE;
    virtual const char *directory() const FL_OVERRIDE;
    virtual void title(const char *t) FL_OVERRIDE;
    virtual const char *title() const FL_OVERRIDE;
    virtual char *filter() const FL_OVERRIDE;
    virtual void filter(const char *f) FL_OVERRIDE;
    virtual int filters() const FL_OVERRIDE;
    virtual void filter_value(int i) FL_OVERRIDE;
    virtual int filter_value() const FL_OVERRIDE;
    virtual void preset_file(const char *f) FL_OVERRIDE;
    virtual const char *preset_file() const FL_OVERRIDE;
    void preset_file(const char *f) FL_OVERRIDE;
    virtual const char *errmsg() const FL_OVERRIDE;
    virtual int show() {return 1;}
};

#endif /*FL_NATIVE_FILE_CHOOSER_H*/
0020 */
0021 #ifndef Fl_Nice_Slider_H
0022 #define Fl_Nice_Slider_H
0023 #include "Fl_Slider.H"
0024
0025 class FL_EXPORT Fl_Nice_Slider : public Fl_Slider {
0026 public:
0027   Fl_Nice_Slider(int X, int Y, int W, int H, const char *L="");
0028   };
0029
0030 #endif

13.96  Fl_Object.H

0002 // Old Fl_Object header file for the Fast Light Tool Kit (FLTK).
0003 //
0004 // Copyright 1998-2010 by Bill Spitzak and others.
0005 //
0006 // This library is free software. Distribution and use rights are outlined in
0007 // the file "COPYING" which should have been included with this file. If this
0008 // file is missing or damaged, see the license at:
0009 //
0010 // https://www.fltk.org/COPYING.php
0011 //
0012 // Please see the following page on how to report bugs and issues:
0013 //
0014 //
0015 //
0016 // This file is provided for back compatibility only. Please use Fl_Widget
0018 #ifndef Fl_Object
0019 #define Fl_Object Fl_Widget
0020 #endif
0021 #include "Fl_Widget.H"

13.97  Fl_Output.H

0002 // Output header file for the Fast Light Tool Kit (FLTK).
0003 //
0004 // Copyright 1998-2022 by Bill Spitzak and others.
0005 //
0006 // This library is free software. Distribution and use rights are outlined in
0007 // the file "COPYING" which should have been included with this file. If this
0008 // file is missing or damaged, see the license at:
0009 //
0010 // https://www.fltk.org/COPYING.php
0011 //
0012 // Please see the following page on how to report bugs and issues:
0013 //
0014 //
0015 //
0016 /* \file
0018   Fl_Output widget . */
0019 #ifndef Fl_Output_H
0020 #define Fl_Output_H
0021 #include "Fl_Input.H"
0022 #include "Fl_Widget.H"
0024 class FL_EXPORT Fl_Output : public Fl_Input {
0026 public:
0027   Fl_Output(int X, int Y, int W, int H, const char *L="");
0028   };
0029 #endif

13.98  Fl_Overlay_Window.H

0002 // Overlay window header file for the Fast Light Tool Kit (FLTK).
0003 //
```cpp
class FL_EXPORT Fl_Overlay_Window : public Fl_Double_Window {
  #ifndef FL_DOXYGEN
  friend class _Fl_Overlay;
  friend class Fl_Window_Driver;
  #endif
  public:
    virtual void draw_overlay() = 0;
    void show() FL_OVERRIDE;
    void hide() FL_OVERRIDE;
    void flush() FL_OVERRIDE;
    void resize(int,int,int,int) FL_OVERRIDE;
    ~Fl_Overlay_Window();
    int can_do_overlay();
    void redraw_overlay();
    protected:
    Fl_Overlay_Window(int W, int H, const char *l=0);
    Fl_Overlay_Window(int X, int Y, int W, int H, const char *l=0);
    public:
    void show(int a, char **b) {Fl_Double_Window::show(a,b);}
    Fl_Overlay_Window *as_overlay_window() FL_OVERRIDE {return this; }
};
```

---

```cpp
class FL_EXPORT Fl_Pack : public Fl_Group {
  int spacing_
  public:
    enum { // values for type(int)
      VERTICAL = 0,
      HORIZONTAL = 1
    };
    protected:
    public:
    enum ( // values for type(int)
      VERTICAL = 0,
      HORIZONTAL = 1
    };
    protected:
    void draw() FL_OVERRIDE;
};
```
13.100 Fl_Paged_Device.H File Reference

declaration of class Fl_Paged_Device.
#include <FL/Fl_Widget_Surface.H>

Classes

- class Fl_Paged_Device
  Represents page-structured drawing surfaces.
- struct Fl_Paged_Device::page_format
  width, height and name of a page format

Macros

- #define NO_PAGE_FORMATS 30 /* MSVC6 compilation fix */
  Number of elements in enum Page_Format.

13.100.1 Detailed Description

declaration of class Fl_Paged_Device.

13.101 Fl_Paged_Device.H

Go to the documentation of this file.
13.102 Fl_Pixmap.H

00049 A2,
00049 A3,
00050 A4,
00051 A5,
00052 A6,
00053 A7,
00054 A8,
00055 A9,
00056 B0,
00057 B1,
00058 B2,
00059 B3,
00060 B4,
00061 B5,
00062 B6,
00063 B7,
00064 B8,
00065 B9,
00066 B10,
00067 C5E,
00068 DLE,
00069 EXECUTIVE,
00070 FOLIO,
00071 LEGDER,
00072 LEGAL,
00073 LETTER,
00074 TABLOID,
00075 ENVELOPE,
00076 MEDIA = 0x1000
00077
00081 enum Page_Layout {
00082 PORTRAIT = 0,
00083 LANDSCAPE = 0x100,
00084 REVERSED = 0x200,
00085 ORIENTATION = 0x300
00086
00087
00088 typedef struct {
00089 int width;
00090 int height;
00091 const char *name;
00092 } page_format;
00093 static const page_format page_formats[NO_PAGE_FORMATS];
00094 virtual ~Fl_Paged_Device() {
00095 virtual int begin_page(void);
00096 virtual void margins(int *left, int *top, int *right, int *bottom);
00097 virtual void scale(float scale_x, float scale_y = 0.);
00098 virtual void rotate(float angle);
00099 void print_widget(Fl_Widget * widget, int delta_x = 0, int delta_y = 0) {draw(widget, delta_x,
00100 delta_y);
00101 void print_window(Fl_Window *win, int x_offset = 0, int y_offset = 0) {
00102 draw_decorated_window(win, x_offset, y_offset);
00103 virtual int end_page (void);
00104 virtual void end_job (void);
00105
00106 #endif // Fl_Paged_Device_H
00107
00108 // Pixmap header file for the Fast Light Tool Kit (FLTK).
00109 // Copyright 1998-2017 by Bill Spitzak and others.
00110 // This library is free software. Distribution and use rights are outlined in
00111 // the file "COPYING" which should have been included with this file. If this
00112 // file is missing or damaged, see the license at:
00113 // https://www.fltk.org/COPYING.php
00114 // Please see the following page on how to report bugs and issues:
00115 // https://www.fltk.org/bugs.php
00116 Generated by Doxygen

13.102 Fl_Pixmap.H
/* FlPixmap widget. */

#ifndef Fl_Pixmap_H
#define Fl_Pixmap_H
#include "Fl_Image.H"

class Fl_Widget;
struct Fl_Menu_Item;

// Older C++ compilers don't support the explicit keyword... :(
#if defined(__sgi) && !defined(_COMPILER_VERSION)
#define explicit
#endif // __sgi && !_COMPILER_VERSION

class FL_EXPORT Fl_Pixmap : public Fl_Image {
friend class Fl_Graphics_Driver;
void copy_data();
void delete_data();
void set_data(const char * const *p);

protected:
void measure();

public:
int alloc_data; // Non-zero if data was allocated

private:
// for internal use
fl_uintptr_t id_;
fl_uintptr_t mask_
int cache_w_, cache_h_; // size of pixmap when cached

public:

explicit Fl_Pixmap(char * const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
{set_data((const char*const*)D); measure();}
explicit Fl_Pixmap(uchar * const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
{set_data((const char*const*)D); measure();}
explicit Fl_Pixmap(const char * const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
{set_data((const char*const*)D); measure();}
explicit Fl_Pixmap(const uchar * const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
{set_data((const char*const*)D); measure();}
virtual ~Fl_Pixmap();

Fl_Image *copy(int W, int H) const FL_OVERRIDE;
Fl_Image *copy() const { return Fl_Image::copy(); }
void color_average(Fl_Color c, float i) FL_OVERRIDE;
void desaturate() FL_OVERRIDE;
draw(int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE;
draw(int X, int Y) {draw(X, Y, W(), H(), 0, 0);} 
void label(Fl_Widget *w) FL_OVERRIDE;
void label(Fl_Menu_Item *m) FL_OVERRIDE;
void uncache() FL_OVERRIDE;
int cache_w() {return cache_w_;
int cache_h() {return cache_h_;}

};

#endif

/* FlPlugin class. */

#ifndef Fl_Plugin_H
#define Fl_Plugin_H
#include "Fl_Preferences.H"
friend class Fl_Graphics_Driver;
void copy_data();
void delete_data();
void set_data(const char * const *p);

protected:
void measure();

public:
int alloc_data; // Non-zero if data was allocated

private:
// for internal use
fl_uintptr_t id_;
fl_uintptr_t mask_
int cache_w_, cache_h_; // size of pixmap when cached

public:

explicit Fl_Pixmap(char * const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
{set_data((const char*const*)D); measure();}
explicit Fl_Pixmap(uchar * const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
{set_data((const char*const*)D); measure();}
explicit Fl_Pixmap(const char * const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
{set_data((const char*const*)D); measure();}
explicit Fl_Pixmap(const uchar * const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
{set_data((const char*const*)D); measure();}
virtual ~Fl_Pixmap();

Fl_Image *copy(int W, int H) const FL_OVERRIDE;
Fl_Image *copy() const { return Fl_Image::copy(); }
void color_average(Fl_Color c, float i) FL_OVERRIDE;
void desaturate() FL_OVERRIDE;
draw(int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE;
draw(int X, int Y) {draw(X, Y, W(), H(), 0, 0);} 
void label(Fl_Widget *w) FL_OVERRIDE;
void label(Fl_Menu_Item *m) FL_OVERRIDE;
void uncache() FL_OVERRIDE;
int cache_w() {return cache_w_;
int cache_h() {return cache_h_;}

};

#endif
class FL_EXPORT Fl_Plugin {
  Fl_Preferences::ID id;

  public:
  Fl_Plugin(const char *klass, const char *name);
  virtual ~Fl_Plugin();
};

class FL_EXPORT Fl_Plugin_Manager : public Fl_Preferences {

  public:
  Fl_Plugin_Manager(const char *klass);
  ~Fl_Plugin_Manager();

  int plugins() { return groups(); }  // return groups()
  Fl_Plugin *plugin(int index);
  Fl_Plugin *plugin(const char *name);
  Fl_Preferences::ID addPlugin(const char *name, Fl_Plugin *plugin);
  
  static void removePlugin(Fl_Preferences::ID id);
  static int load(const char *filename);
  static int loadAll(const char *dirpath, const char *pattern=0);
};

#endif // !Fl_Preferences_H

_Delete_ 

---

class FL_EXPORT Fl_PNG_Image { public Fl_RGB_Image {

  public:
  Fl_PNG_Image(const char * filename);
  Fl_PNG_Image(const char *name_png, const unsigned char *buffer, int datasize);

  private:
  Fl_PNG_Image(const char *filename, int offset);  // used by Fl_ICO_Image

  void load_png_(const char *name_png, int offset, const unsigned char *buffer_png, int datasize);
};

// Support functions to write PNG image files (since 1.4.0)

FL_EXPORT int fl_write_png(const char *filename, Fl_RGB_Image *img);
FL_EXPORT int fl_write_png(const char *filename, const char *pixels, int w, int h, int d=3, int ld=0);
FL_EXPORT int fl_write_png(const char *filename, const unsigned char *buffer_png, int datasize);

#endif

---

- Fl_PNM_Image.H

// PNM image header file for the Fast Light Tool Kit (FLTK).

#ifndef Fl_PNM_Image_H
#define Fl_PNM_Image_H

#include "Fl_RGB_Image.H"

class FL_EXPORT Fl_PNM_Image { public Fl_RGB_Image {

  friend class Fl_ICO_Image;

  public:

  Fl_PNM_Image(const char * filename);

  Fl_PNM_Image(const char *name_png, const unsigned char *buffer, int datasize);

  private:

  Fl_PNM_Image(const char *filename, int offset);  // used by Fl_ICO_Image

  void load_png_(const char *name_png, int offset, const unsigned char *buffer_png, int datasize);

  };;

#endif

---

- Fl_PNG_Image.H

// PNG image header file for the Fast Light Tool Kit (FLTK).

#ifndef Fl_PNG_Image_H
#define Fl_PNG_Image_H

#include "Fl_RGB_Image.H"

class FL_EXPORT Fl_PNG_Image { public Fl_RGB_Image {

  friend class Fl_ICO_Image;

  public:

  Fl_PNG_Image(const char * filename);

  Fl_PNG_Image(const char *name_png, const unsigned char *buffer, int datasize);

  private:

  Fl_PNG_Image(const char *filename, int offset);  // used by Fl_ICO_Image

  void load_png_(const char *name_png, int offset, const unsigned char *buffer_png, int datasize);

  };;

#endif

---

- Fl_PNM_Image.H

// PNM image header file for the Fast Light Tool Kit (FLTK).

#ifndef Fl_PNM_Image_H
#define Fl_PNM_Image_H

#include "Fl_RGB_Image.H"

class FL_EXPORT Fl_PNM_Image { public Fl_RGB_Image {

  friend class Fl_ICO_Image;

  public:

  Fl_PNM_Image(const char * filename);

  Fl_PNM_Image(const char *name_png, const unsigned char *buffer, int datasize);

  private:

  Fl_PNM_Image(const char *filename, int offset);  // used by Fl_ICO_Image

  void load_png_(const char *name_png, int offset, const unsigned char *buffer_png, int datasize);

  };;

#endif

---

Generated by Doxygen
```cpp
/* Fl_PNM_Image class */

#ifndef Fl_PNM_Image_H
#define Fl_PNM_Image_H
#include "Fl_RGB_Image.H"

class FL_EXPORT Fl_PNM_Image : public Fl_RGB_Image {
  public:
    Fl_PNM_Image(const char * filename);
};
#endif
```

13.106 Fl_Positioner.H

```cpp
/* Fl_Positioner widget */

#ifndef Fl_Positioner_H
#define Fl_Positioner_H
#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

class FL_EXPORT Fl_Positioner : public Fl_Widget {
  double xmin, ymin;
  double xmax, ymax;
  double xvalue_, yvalue_;  // values to display
  double xstep_, ystep_;  // step sizes for %change

  protected:
    // these allow subclasses to put the dial in a smaller area:
    void draw(int, int, int, int);
    int handle(int, int, int, int, int);
  public:
    int handle(int) FL_OVERRIDE;
    Fl_Positioner(int x, int y, int w, int h, const char *l=0);
    double xvalue() const {return xvalue_;
    int xvalue(double);
    int yvalue(double);
    int value(double, double);
    void xbounds(double, double);
    double xminimum() const {return xmin;
    void xminimum(double a) {xbounds(a, xmax);
    double xmaximum() const {return xmax;
    void xmaximum(double a) {xbounds(xmin, a);
    void ybounds(double, double);
    double yminimum() const {return ymin;
    void yminimum(double a) {ybounds(a, ymax);
    double ymaximum() const {return ymax;
    }
};
#endif
```
### 13.107 Fl_PostScript.H File Reference

Declaration of classes `Fl_PostScript_File_Device` and `Fl_EPS_File_Surface`.

```
#include <FL/Fl_Paged_Device.H>
#include <FL/fl_draw.H>
#include <stdarg.h>
```

#### Classes

- **class Fl_EPS_File_Surface**
  
  Encapsulated PostScript drawing surface.

- **class Fl_PostScript_File_Device**
  
  To send graphical output to a PostScript file.

#### Typedefs

- **typedef int(* Fl_PostScript_Close_Command) (FILE *)**
  
  Signature of functions FLTK may use to close FILE variables after PostScript/EPS output.

### 13.107.1 Detailed Description

Declaration of classes `Fl_PostScript_File_Device` and `Fl_EPS_File_Surface`.

### 13.107.2 Typedef Documentation

#### 13.107.2.1 Fl_PostScript_Close_Command

**typedef int(* Fl_PostScript_Close_Command) (FILE *)**

Signature of functions FLTK may use to close FILE variables after PostScript/EPS output.

A non-null return value indicates output error.

**See also**

`Fl_PostScript_File_Device::close_command()` and `Fl_EPS_File_Surface::Fl_EPS_File_Surface()`.

### 13.108 Fl_PostScript.H

Go to the documentation of this file.
```cpp
#include <FL/fl_draw.H>
#include <stdarg.h>

extern "C" {
  typedef int (Fl_PostScript_Close_Command)(FILE *);
}

class Fl_PostScript_Graphics_Driver;

class FL_EXPORT Fl_PostScript_File_Device : public Fl_Paged_Device {
  private:
    // memorize the display's current font to restore it when the object ceases being current
  Fl_Font display_font_{
  Fl_Fontsize display_size_{
  protected:
    inline Fl_PostScript_Graphics_Driver *driver() { return (Fl_PostScript_Graphics_Driver*)Fl_Surface_Device::driver(); }

  public:
    Fl_PostScript_File_Device(){
    Fl_PostScript_File_Device();

    int begin_job(int pagecount, int from, int to, char *perr_message) FL_OVERRIDE;
  
    int begin_job(int pagecount = 0, enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
                  enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT);  
    int start_job(int pagecount = 0, enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
                  enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT) {
      return begin_job(pagecount, format, layout);
    }
    int begin_job(FILE *ps_output, int pagecount = 0, enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
                  enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT) {
      return begin_job(ps_output, pagecount, format, layout);
    }

    int begin_page () FL_OVERRIDE;
  
    int printable_rect(int *w, int *h) FL_OVERRIDE;
  
    void margins(int *left, int *top, int *right, int *bottom) FL_OVERRIDE;
  
    void origin(int *x, int *y) FL_OVERRIDE;
  
    void scale (float scale_x, float scale_y = 0.) FL_OVERRIDE;
  
    void rotate(float angle) FL_OVERRIDE;
  
    void translate(int x, int y) FL_OVERRIDE;
  
    void untranslate(void) FL_OVERRIDE;
  
    int end_page (void) FL_OVERRIDE;
  
    int end_job(void) FL_OVERRIDE;
  
    static const char *file_chooser_title;
  
    FILE *file();
  
    FILE *file();
  
    void close_command(Fl_PostScript_Close_Command cmd);
  
    void set_current() FL_OVERRIDE;
  
    void end_current() FL_OVERRIDE;
  
  
  
  
  
  
  class FL_EXPORT Fl_EPS_File_Surface : public Fl_Widget_Surface {
    protected:
      inline Fl_PostScript_Graphics_Driver *driver() { return (Fl_PostScript_Graphics_Driver*)Fl_Surface_Device::driver(); }

    public:
      Fl_EPS_File_Surface(int width, int height, FILE *eps_output,
                          Fl_Color background = FL_WHITE, Fl_PostScript_Close_Command closef = NULL);
      ~Fl_EPS_File_Surface();

      int printable_rect(int *w, int *h) FL_OVERRIDE;

      FILE *file();

      void origin(int *x, int *y) FL_OVERRIDE;

      void origin(int *px, int *py) FL_OVERRIDE;

      void translate(int *x, int *y) FL_OVERRIDE;

      void untranslate() FL_OVERRIDE;

      int close();

    
  
  
  
  
  
  #endif // Fl_PostScript_H
```

13.109 Fl_Preferences.H

// Preferences implementation for the Fast Light Tool Kit (FLTK).

// Copyright 2002-2023 by Matthias Melcher.

// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:

// https://www.fltk.org/COPYING.php

Generated by Doxygen
Please see the following page on how to report bugs and issues:
https://www.fltk.org/bugs.php

```c
/*
 ** Fl_Preferences class . */

#ifndef Fl_Preferences_H
#define Fl_Preferences_H

#include <stdio.h>
#include "Fl_Export.H"
#include "fl_attr.h"

//class Fl_String;
#if (FLTK_USE_STD)
#include <string>
#endif

class FL_EXPORT Fl_Preferences {

public:
    enum Root {
        UNKNOWN_ROOT_TYPE = -1,
        SYSTEM = 0,
        USER,
        MEMORY,
        ROOT_MASK = 0x00FF,
        CORE = 0x0100,
        C_LOCALE = 0x0100,
        CLEAR = 0x2000,
        SYSTEM_L = SYSTEM | C_LOCALE,
        USER_L = USER | C_LOCALE,
        CORE_SYSTEM_L = CORE | SYSTEM_L,
        CORE_USER_L = CORE | USER_L,
        CORE_SYSTEM = CORE | SYSTEM,
        CORE_USER = CORE | USER
    };

    typedef void *ID;

    static const char *new_UUID();

    static const unsigned int NONE = 0x0000;
    static const unsigned int USER_READ_OK = 0x0001;
    static const unsigned int USER_WRITE_OK = 0x0002;
    static const unsigned int USER_OK = USER_READ_OK | USER_WRITE_OK;
    static const unsigned int SYSTEM_READ_OK = 0x0004;
    static const unsigned int SYSTEM_WRITE_OK = 0x0008;
    static const unsigned int SYSTEM_OK = SYSTEM_READ_OK | SYSTEM_WRITE_OK;
    static const unsigned int APP_OK = SYSTEM_OK | USER_OK;
    static const unsigned int CORE_READ_OK = 0x0010;
    static const unsigned int CORE_WRITE_OK = 0x0020;
    static const unsigned int CORE_OK = CORE_READ_OK | CORE_WRITE_OK;
    static const unsigned int ALL_READ_OK = USER_READ_OK | SYSTEM_READ_OK | CORE_READ_OK;
    static const unsigned int ALL_WRITE_OK = USER_WRITE_OK | SYSTEM_WRITE_OK | CORE_WRITE_OK;
    static const unsigned int ALL = ALL_READ_OK | ALL_WRITE_OK;

    static void file_access(unsigned int flags);
    static unsigned int file_access();
    static Root filename( char *buffer, size_t buffer_size, Root root, const char *vendor, const char *application );

    Fl_Preferences( Root root, const char *vendor, const char *application );
    Fl_Preferences( const char *path, const char *vendor, const char *application, Root flags );
    Fl_Preferences( Fl_Preferences &parent, const char *group );
    Fl_Preferences( Fl_Preferences *parent, const char *group );
    Fl_Preferences( Fl_Preferences &parent, int groupIndex );
    Fl_Preferences( Fl_Preferences *parent, int groupIndex );
    Fl_Preferences(const Fl_Preferences&);
    virtual ~Fl_Preferences();

    static void file_access(unsigned int flags);
    static unsigned int file_access();

    static Root filename( char *buffer, size_t buffer_size, Root root, const char *vendor, const char *application );
    Fl_Preferences( Root root, const char *vendor, const char *application );
    Fl_Preferences( const char *path, const char *vendor, const char *application, Root flags );
    Fl_Preferences( Fl_Preferences &parent, const char *group );
    Fl_Preferences( Fl_Preferences &parent, int groupIndex );
    Fl_Preferences(const Fl_Preferences&);
    virtual ~Fl_Preferences();

    Fl_Preferences( const char *path, const char *vendor, const char *application ) ;

    Root filename( char *buffer, size_t buffer_size);

    ID id() { return (ID)node; }

    static char remove(ID id_) { return ((Node *)id_)->remove(); }

    const char *name() { return node->name(); }

    const char *path() { return node->path(); }

    static char remove(ID id_); /* to be removed */

};

Generated by Doxygen
```
int groups();
const char *group( int num_group );
char delete_group( const char *group );
char delete_all_groups();
int entries();
const char *entry( int index );
char entry_exists( const char *key );
char delete_entry( const char *entry );
char delete_all_entries();
char clear();

int entries();
const char *entry( int index );
char entry_exists( const char *key );
char delete_entry( const char *entry );
char delete_all_entries();
char clear();

char set( const char *entry, int value );
char set( const char *entry, float value );
char set( const char *entry, float value, int precision );
char set( const char *entry, double value );
char set( const char *entry, double value, int precision );
char set( const char *entry, const void *value, int size );

char get( const char *entry, int &value, int defaultValue );
char get( const char *entry, float &value, float defaultValue );
char get( const char *entry, double &value, double defaultValue );
char get( const char *entry, char *&value, const char *defaultValue );
char get( const char *entry, char *value, const char *defaultValue, int maxSize );
char get( const char *entry, void *&value, const void *defaultValue, int defaultSize );
char get( const char *entry, void *value, const void *defaultValue, int defaultSize, int maxSize );

// char set( const char *entry, const Fl_String &value );
// char get( const char *entry, Fl_String &value, const Fl_String &defaultValue );

#if (FLTK_USE_STD)
char set( const char *entry, const std::string &value );
char get( const char *entry, std::string &value, const std::string &defaultValue );
#endif

int size( const char *entry );
char get_userdata_path( char *path, int pathlen );
int flush();
int dirty();

static const char *newUUID() { return new_UUID(); }
char groupExists( const char *key ) { return group_exists(key); }
char deleteGroup( const char *group ) { return delete_group(group); }
char deleteAllGroups() { return delete_all_groups(); }
char entryExists( const char *key ) { return entry_exists(key); }
char deleteEntry( const char *entry ) { return delete_entry(entry); }
char deleteAllEntries() { return delete_all_entries(); }
char getUserdataPath( char *path, int pathlen ) { return get_userdata_path(path, pathlen); }

class FL_EXPORT Name {
    char *data_{
    public:
        Name( unsigned int n );
        Name( const char *format, ... );
        ~Name();
        operator const char *();
    private:
        struct Entry {
            char *name, *value;
        };
    };
Node *first_child_, *next_;  
union {  
    Node *parent_;  // top_ bit clear  
    RootNode *root_node_;  // top_ bit set  
};  
char *path_;  
Entry *entry_;  
nint_nEntry_, NEntry_;  
unsigned char dirty_,1;  
unsigned char top_;1;  
unsigned char indexed_,1;  
// indexing routines  
Node *index_;  
nIndex_, NIndex_;  
void createIndex();  
void updateIndex();  
void deleteIndex();  
public:  
    static int lastEntrySet;  
    public:  
    Node( const char *path );  
    ~Node();  
    // node methods  
    int write( FILE *f );  
    const char *name();  
    const char *path() { return path_; }  
    Node *find( const char *path );  
    Node *search( const char *path, int offset=0 );  
    Node *childNode( int ix );  
    void setParent( Node *parent );  
    Node *parent() { return top_?0L:parent_; }  
    Node *findRoot();  
    char remove();  
    char dirty();  
    void clearDirtyFlags();  
    void deleteAllChildren();  
    // entry methods  
    int nChildren();  
    const char *child( int ix );  
    void set( const char *name, const char *value );  
    void set( const char *line );  
    void add( const char *line );  
    const char *get( const char *name );  
    const char *getEntry( const char *name );  
    char deleteEntry( const char *name );  
    void deleteAllEntries();  
    int nEntry() { return nEntry_; }  
    Entry &entry(int i) { return entry_[i]; }  
    };  
};  
friend class Node;  
};  
};  
ья class FL_EXPORT RootNode {  
    Fl_Preferences *prefs_;  
    char *filename_;  
    char *vendor_, *application_;  
    Root root_type_;  
    public:  
    RootNode( Fl_Preferences *, Root root, const char *vendor, const char *application );  
    RootNode( Fl_Preferences *, const char *path, const char *vendor, const char *application, Root flags );  
    RootNode( Fl_Preferences * );  
    ~RootNode();  
    int read();  
    int write();  
    char getPath( char *path, int pathlen );  
    char *filename() { return filename_; }  
    Root root() { return root_type_; }  
    };  
friend class RootNode;  
protected:  
    Node *node;  
    RootNode *rootNode;  
};  
#endif // !Fl_Preferences_H

13.110 Fl_Printer.H File Reference

declaration of class Fl_Printer.


```cpp
#include <FL/Fl_Paged_Device.H>

Classes

• class Fl_Printer

  OS-independent print support.

13.110.1 Detailed Description

declaration of class Fl_Printer.

13.111 Fl_Printer.H

Go to the documentation of this file.

00001 //
00002 // Printing support for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 2010-2016 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file. If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 // https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 // https://www.fltk.org/bugs.php
00015 //
00016 00021 #ifndef Fl_Printer_H
00022 #define Fl_Printer_H
00023
00104 class FL_EXPORT Fl_Printer : public Fl_Paged_Device {
00105 private:
00106   Fl_Paged_Device *printer;
00108   static Fl_Paged_Device * newPrinterDriver(void);
00111   Fl_Printer(void);
00112   int begin_job(int pagecount = 0, int *frompage = NULL, int *topage = NULL, char **perr_message = NULL) FL_OVERRIDE;
00115   int printable_rect(int *w, int *h) FL_OVERRIDE;
00114   void margins(int *left, int *top, int *right, int *bottom) FL_OVERRIDE;
00116   void origin(int *x, int *y) FL_OVERRIDE;
00117   void origin(int x, int y) FL_OVERRIDE;
00118   void scale(float scale_x, float scale_y = 0.) FL_OVERRIDE;
00119   void rotate(float angle) FL_OVERRIDE;
00120   void translate(int x, int y) FL_OVERRIDE;
00121   void untranslate(void) FL_OVERRIDE;
00122   int end_page (void) FL_OVERRIDE;
00123   void end_job (void) FL_OVERRIDE;
00124   void set_current(void) FL_OVERRIDE;
00125   bool is_current() FL_OVERRIDE;
00126   00130 static const char *dialog_title;
00131 static const char *dialog_printer;
00132 static const char *dialog_range;
00133 static const char *dialog_copies;
00134 static const char *dialog_all;
00135 static const char *dialog_pages;
00136 static const char *dialog_from;
00137 static const char *dialog_to;
00138 static const char *dialog_properties;
00139 static const char *dialog_print_button;
00140 static const char *dialog_cancel_button;
00141 static const char *dialog_print_to_file;
00142 static const char *property_title;
00143 static const char *property_pagesize;
00144 static const char *property_mode;
00145 static const char *property_use;
00146 static const char *property_save;
00147 static const char *property_cancel;
00151 ~Fl_Printer(void);
00152

```

Generated by Doxygen
13.12 Fl_Progress.H

```
0001 //
0002 // Progress bar widget definitions.
0003 //
0004 // Copyright 2000-2010 by Michael Sweet.
0005 //
0006 // This library is free software. Distribution and use rights are outlined in
0007 // the file *COPYING* which should have been included with this file. If this
0008 // file is missing or damaged, see the license at:
0009 //
0010 // https://www.fltk.org/COPYING.php
0011 //
0012 // Please see the following page on how to report bugs and issues:
0013 //
0014 // https://www.fltk.org/bugs.php
0015 //
0016 #ifndef _Fl_Progress_H_
0017 # define _Fl_Progress_H_
0018
0019 //
0020 // Include necessary headers.
0021 //
0022 #include "Fl_Widget.H"
0023
0024 //
0025 //
0026 #ifndef __Fl_Progress_H
0027 #include "Fl_Widget.H"
0028
0029 #endif
0030 //
0031 // Progress class...
0032 //
0033 class FL_EXPORT Fl_Progress : public Fl_Widget {
0034
0035 float value_,
0036 minimum_,
0037 maximum_;
0038
0039 protected:
0040
0041 void draw() FL_OVERRIDE;
0042
0043 public:
0044
0045 Fl_Progress(int x, int y, int w, int h, const char *l = 0);
0046
0047 float maximum(float v) { maximum_ = v; redraw(); }   
0048 float minimum(float v) { minimum_ = v; redraw(); }   
0049 float value() const { return (value_); }           
0050 void value(float v) { value_ = v; redraw(); }        
0051 void maximum(float v) { maximum_ = v; redraw(); }    
0052 void minimum(float v) { minimum_ = v; redraw(); }    
0053
0054 #endif // !_Fl_Progress_H_
```

13.13 Fl_Radio_Button.H

```
0001 //
0002 // Radio button header file for the Fast Light Tool Kit (FLTK).
0003 //
0004 // Copyright 1998-2014 by Bill Spitzak and others.
0005 //
0006 // This library is free software. Distribution and use rights are outlined in
0007 // the file *COPYING* which should have been included with this file. If this
0008 // file is missing or damaged, see the license at:
0009 //
0010 // https://www.fltk.org/COPYING.php
0011 //
0012 // Please see the following page on how to report bugs and issues:
0013 //
0014 // https://www.fltk.org/bugs.php
0015 //
0016 #ifndef _Fl_Radio_Button_H_
0017 # define _Fl_Radio_Button_H_
0018
0019 //
0020 // Include necessary headers.
0021 //
0022 #include "Fl_Widget.H"
0023
0024 //
0025 //
0026 #ifndef __Fl_Radio_Button_H
0027 #include "Fl_Widget.H"
0028
0029 #endif
0030 //
0031 // Radio button class...
0032 //
0033 class FL_EXPORT Fl_Radio_Button : public Fl_Widget {
0034
0035 public:
0036
0037 Fl_Radio_Button(int x, int y, int w, int h, const char *l = 0);
0038
0039 virtual ~Fl_Radio_Button();
0040
0041 float on() const { return (on_); }
0042 void on(float v) { on_ = v; }
0043
0044 void set_label(const char *l) { label_ = l; }
0045
0046 void status(int s) { status_ = s; }   
0047 void status(int s) const { return (status_); }   
0048
0049 #endif // !_Fl_Radio_Button_H_
```
```cpp
2124 #ifndef Fl_Radio_Button_H
2125 #define Fl_Radio_Button_H
2126
2127 #include "Fl_Button.H"
2128
2129 class FL_EXPORT Fl_Radio_Button : public Fl_Button {
2130 public:
2131 Fl_Radio_Button(int X,int Y,int W,int H,const char *L=0);
2132 };  
2133 #endif

13.114 Fl_Radio_Light_Button.H

```cpp
2134 #ifndef Fl_Radio_Light_Button_H
2135 #define Fl_Radio_Light_Button_H
2136
2137 #include "Fl_Light_Button.H"
2138
2139 class FL_EXPORT Fl_Radio_Light_Button : public Fl_Light_Button {
2140 public:
2141 Fl_Radio_Light_Button(int X,int Y,int W,int H,const char *l=0);
2142 };  
2143 #endif

13.115 Fl_Radio_Round_Button.H

```cpp
2144 #ifndef Fl_Radio_Round_Button_H
2145 #define Fl_Radio_Round_Button_H
2146
2147 #include "Fl_Round_Button.H"
2148
2149 class FL_EXPORT Fl_Radio_Round_Button : public Fl_Round_Button {
2150 public:
2151 Fl_Radio_Round_Button(int X,int Y,int W,int H,const char *L=0);
2152 };  
2153 #endif
```
ifndef Fl_Rect_H
#define Fl_Rect_H

#include <FL/Fl_Widget.H> // for c’tor based on Fl_Widget

class FL_EXPORT Fl_Rect {

  int x_;  // top-left x of box
  int y_;  // top-left y of box
  int w_;  // width of box
  int h_;  // height of box

public:

  Fl_Rect();               // default constructor
  Fl_Rect(int W, int H);  // width and height
  Fl_Rect(int X, int Y, int W, int H); // position and size
  Fl_Rect(int X, int Y, int W, int H, Fl_Boxtype bt); // position, size, and box type
  Fl_Rect(const Fl_Widget& widget); // constructor from Fl_Widget
  Fl_Rect(const Fl_Widget* const widget); // constructor from Fl_Widget pointer

  int x() const { return x_; } // x coordinate
  int y() const { return y_; } // y coordinate
  int w() const { return w_; } // width
  int h() const { return h_; } // height

  int r() const { return x_ + w_; } // right edge
  int b() const { return y_ + h_; } // bottom edge

  void x(int X) { x_ = X; } // set x
  void y(int Y) { y_ = Y; } // set y
  void w(int W) { w_ = W; } // set width
  void h(int H) { h_ = H; } // set height

  void inset(int d) { // inset all sides by d
    x_ += d;
    y_ += d;
    w_ -= 2 * d;
    h_ -= 2 * d;
  }

  void inset(Fl_Boxtype bt) { // inset all sides by Fl::box_dx(bt)
    x_ += Fl::box_dx(bt);
    y_ += Fl::box_dy(bt);
    w_ -= Fl::box_dw(bt);
    h_ -= Fl::box_dh(bt);
  }

  void inset(int left, int top, int right, int bottom) { // inset by left, top, right, bottom
    x_ += left;
    y_ += top;
    w_ -= (left + right);
    h_ -= (top + bottom);
  }

};

#edef Fl_Rect_H
friend bool operator==(const Fl_Rect& lhs, const Fl_Rect& rhs) {
  return (lhs.x_==rhs.x_) && (lhs.y_==rhs.y_) && (lhs.w_==rhs.w_) && (lhs.h_==rhs.h_);
}

friend bool operator!=(const Fl_Rect& lhs, const Fl_Rect& rhs) {
  return !(lhs==rhs);
}

}; // class Fl_Rect

13.117 Fl_Repeat_Button.H

/*
  file
Fl_Repeat_Button widget . */

#ifndef Fl_Repeat_Button_H
#define Fl_Repeat_Button_H
#include "Fl.H"
#include "Fl_Button.H"

class FL_EXPORT Fl_Repeat_Button : public Fl_Button {
  static void repeat_callback(void *);
  public:
    int handle(int) FL_OVERRIDE;
    Fl_Repeat_Button(int X,int Y,int W,int H,const char *l=0);
    void deactivate() {
      Fl::remove_timeout(repeat_callback,this);
      Fl_Button::deactivate();
    }
  }
#endif

13.118 Fl_Return_Button.H

/*
  file
Fl_Return_Button widget . */

#ifndef Fl_Return_Button_H
#define Fl_Return_Button_H
#include "Fl_Button.H"

class FL_EXPORT Fl_Return_Button : public Fl_Button {
  protected:
    void draw() FL_OVERRIDE;
    public:
    Fl_Return_Button(int X,int Y,int W,int H,char *l=0);
    void deactivate() {
      Fl::remove_timeout(repeat_callback,this);
      Fl_Button::deactivate();
    }
  }
#endif

Generated by Doxygen
13.119  Fl_RGB_Image.H

00034  public:
00035     int handle(int) FL_OVERRIDE;
00036     Fl_Return_Button(int X, int Y, int W, int H, const char *l=0);
00037  };
00038 #endif

13.119  Fl_RGB_Image.H

00001  //
00002  // RGB Image header file for the Fast Light Tool Kit (FLTK).
00003  //
00004  // Copyright 1998-2010 by Bill Spitzak and others.
00005  //
00006  // This library is free software. Distribution and use rights are outlined in
00007  // the file 'COPYING' which should have been included with this file. If this
00008  // file is missing or damaged, see the license at:
00009  //
00010  //  https://www.fltk.org/COPYING.php
00011  //
00012  // Please see the following page on how to report bugs and issues:
00013  //
00014  //  https://www.fltk.org/bugs.php
00015  //
00016  #ifndef Fl_RGB_Image_H
00017  # define Fl_RGB_Image_H
00018  # include "Fl_Image.H"
00019  # include "Fl_Valuator.H"
00020  #endif // Fl_RGB_Image_H

13.120  Fl_Roller.H

00001  //
00002  // Roller header file for the Fast Light Tool Kit (FLTK).
00003  //
00004  // Copyright 1998-2023 by Bill Spitzak and others.
00005  //
00006  // This library is free software. Distribution and use rights are outlined in
00007  // the file 'COPYING' which should have been included with this file. If this
00008  // file is missing or damaged, see the license at:
00009  //
00010  //  https://www.fltk.org/COPYING.php
00011  //
00012  // Please see the following page on how to report bugs and issues:
00013  //
00014  //  https://www.fltk.org/bugs.php
00015  //
00016  #ifndef Fl_Roller_H
00017  #define Fl_Roller_H
00018  
00019  #ifndef Fl_Valuator_H
00020  #include "Fl_Valuator.H"
00021  #define Fl_Roller_H
00022  #endif
00023  #ifndef Fl_Valuator_H
00024  #include "Fl_Valuator.H"
00025  #endif
00026  
00027  class FL_EXPORT Fl_Roller : public Fl_Valuator {
00028     protected:
00029     virtual void draw() FL_OVERRIDE;
00030     int handle(int) FL_OVERRIDE;
00031     Fl_Roller(int X, int Y, int W, int H, const char *l=0);
00032  };
00033 #endif

13.121  Fl_Round_Button.H

00001  //
00002  // Round button header file for the Fast Light Tool Kit (FLTK).
00003  //
00004  // Copyright 1998-2022 by Bill Spitzak and others.
00005  //
00006  // This library is free software. Distribution and use rights are outlined in
00007  // the file 'COPYING' which should have been included with this file. If this
00008  // file is missing or damaged, see the license at:
00009  //
00010  //  https://www.fltk.org/COPYING.php
00011  //
00012  // Please see the following page on how to report bugs and issues:
00013  //
00014  //  https://www.fltk.org/bugs.php
00015  //
00016  */
00017  Fl_Roller widget . */
00018  
00019  #ifndef Fl_Roller_H
00020  #define Fl_Roller_H
00021  #define Fl_Roller_H
00022  #endif
00023  #ifndef Fl_Valuator_H
00024  #include "Fl_Valuator.H"
00025  #endif
00026  
00027  class FL_EXPORT Fl_Round_Button : public Fl_Valuator {
00028     protected:
00029     virtual void draw() FL_OVERRIDE;
00030     int handle(int) FL_OVERRIDE;
00031     Fl_Round_Button(int X, int Y, int W, int H, const char *l=0);
00032  };
00033 #endif

Generated by Doxygen
```cpp
#ifndef Fl_Round_Button_H
#define Fl_Round_Button_H

#include "Fl_Light_Button.H"

class FL_EXPORT Fl_Round_Button : public Fl_Light_Button {
public:
    Fl_Round_Button(int x, int y, int w, int h, const char *l = 0);
};

#endif

13.122 Fl_Round_Clock.H

#ifndef Fl_Round_Clock_H
#define Fl_Round_Clock_H

#include "Fl_Clock.H"

class FL_EXPORT Fl_Round_Clock : public Fl_Clock {
public:
    Fl_Round_Clock(int X, int Y, int W, int H, const char *L = 0);
};

#endif

13.123 Fl_Scheme.H

#ifndef FL_Fl_Scheme_H_
#define FL_Fl_Scheme_H_

#include <FL/Fl.H>

class Fl_Scheme {
private:
    static const char **names_; // registered scheme names
    static int num_schemes_; // number of registered schemes
    static int alloc_size_; // number of allocated scheme name entries
};
```
13.124 Fl_Scheme_Choice.H

protected:

// const char *name_; // the scheme’s name

// protected constructor - not yet implemented

// Fl_Scheme(const char *name);

public:

// Static methods.

// Some of these methods will replace the scheme related methods of class Fl,
// for instance Fl::scheme() and Fl::is_scheme().
// Backwards compatibility must be kept though.

static const char **names();

static int num_schemes() {
    if (!names_) names(); // force initialization
    return num_schemes_
}

static int add_scheme_name(const char *name);

}; // class Fl_Scheme

#endif // FL_Fl_Scheme_H_

13.125 Fl_Scroll.H

protected:

static void scheme_cb_(Fl_Widget *w, void *);

public:

Fl_Scheme_Choice(int X, int Y, int W, int H, const char *L = 0);

int handle(int event) FL_OVERRIDE;

virtual void init_value();

}; // class Fl_Scheme_Choice

#endif // FL_Fl_Scheme_Choice_H_
/*
 * Fl_Scroll widget . */

#ifndef Fl_Scroll_H
#define Fl_Scroll_H

#include "Fl_Group.H"
#include "Fl_Scrollbar.H"

class FL_EXPORT Fl_Scroll : public Fl_Group {

int xposition_, yposition_;  // (STR#1895)
int oldx, oldy;
int scrollbar_size_;  // (STR#1895)
static void hscrollbar_cb(Fl_Widget *, void*);
static void scrollbar_cb(Fl_Widget *, void*);
static void draw_clip(void *,int,int,int,int);

protected:

typedef struct { int x,y,w,h; } Fl_Region_XYWH;

typedef struct { int l; int r; int t; int b; } Fl_Region_LRTB;

typedef struct { int x,y,w,h; int pos; int size; int first; int total; } Fl_Scrollbar_Data;

typedef struct { int scrollsize
 Fl_Region_XYWH innerbox;
 Fl_Region_XYWH innerchild;
 Fl_Region_LRTB child;
 int hneeded;
 int vneeded;
 Fl_Scrollbar_Data hscroll;
 Fl_Scrollbar_Data vscroll;
	} ScrollInfo;

void recalc_scrollbars(ScrollInfo &si) const;

int on_insert(Fl_Widget *, int) FL_OVERRIDE;
int on_move(int, int) FL_OVERRIDE;
void fix_scrollbar_order();
void bbox(int&,int&,int&,int&) const;
void draw() FL_OVERRIDE;

public:

Fl_Scrollbar scrollbar;
Fl_Scrollbar hscrollbar;

void resize(int X, int Y, int W, int H) FL_OVERRIDE;
handle(int) FL_OVERRIDE;

int Fl_Scroll::type() const {
enum { // values for type()
HORIZONTAL = 1,
VERTICAL = 2,
BOTH = 3,
ALWAYS_ON = 4,
HORIZONTAL_ALWAYS = 5,
VERTICAL_ALWAYS = 6,
BOTH ALWAYS = 7
};

Generated by Doxygen
```c++
int xposition() const { return xposition_; }

int yposition() const { return yposition_; }

void scroll_to(int, int);

void clear();

// delete child n (by index)
int delete_child(int n) FL_OVERRIDE;

int scrollbar_size() const {
  return (scrollbar_size_);
}

void scrollbar_size(int newSize) {
  if (newSize != scrollbar_size_) redraw();
  scrollbar_size_ = newSize;
}

Fl_Scrollbar::Fl_Scrollbar(int X, int Y, int W, int H, const char *L = 0);

~Fl_Scrollbar();

int handle(int) FL_OVERRIDE;

int linesize() const { return linesize_; }

void linesize(int i) { linesize_ = i; }
```

### Fl_Scrollbar.H

```c++
int xposition() const { return xposition_; }

int yposition() const { return yposition_; }

void scroll_to(int, int);

void clear();

// delete child n (by index)
int delete_child(int n) FL_OVERRIDE;

int scrollbar_size() const {
  return (scrollbar_size_);
}

void scrollbar_size(int newSize) {
  if (newSize != scrollbar_size_) redraw();
  scrollbar_size_ = newSize;
}

Fl_Scrollbar::Fl_Scrollbar(int X, int Y, int W, int H, const char *L = 0);

~Fl_Scrollbar();

int handle(int) FL_OVERRIDE;

int linesize() const { return linesize_; }

void linesize(int i) { linesize_ = i; }
```

### Fl_Secret_Input.H

```c++
int xposition() const { return xposition_; }

int yposition() const { return yposition_; }

void scroll_to(int, int);

void clear();

// delete child n (by index)
int delete_child(int n) FL_OVERRIDE;

int scrollbar_size() const {
  return (scrollbar_size_);
}

void scrollbar_size(int newSize) {
  if (newSize != scrollbar_size_) redraw();
  scrollbar_size_ = newSize;
}

Fl_Scrollbar::Fl_Scrollbar(int X, int Y, int W, int H, const char *L = 0);

~Fl_Scrollbar();

int handle(int) FL_OVERRIDE;

int linesize() const { return linesize_; }

void linesize(int i) { linesize_ = i; }
```

```c++
int xposition() const { return xposition_; }

int yposition() const { return yposition_; }

void scroll_to(int, int);

void clear();

// delete child n (by index)
int delete_child(int n) FL_OVERRIDE;

int scrollbar_size() const {
  return (scrollbar_size_);
}

void scrollbar_size(int newSize) {
  if (newSize != scrollbar_size_) redraw();
  scrollbar_size_ = newSize;
}

Fl_Scrollbar::Fl_Scrollbar(int X, int Y, int W, int H, const char *L = 0);

~Fl_Scrollbar();

int handle(int) FL_OVERRIDE;

int linesize() const { return linesize_; }

void linesize(int i) { linesize_ = i; }
```

```c++
int xposition() const { return xposition_; }

int yposition() const { return yposition_; }

void scroll_to(int, int);

void clear();

// delete child n (by index)
int delete_child(int n) FL_OVERRIDE;

int scrollbar_size() const {
  return (scrollbar_size_);
}

void scrollbar_size(int newSize) {
  if (newSize != scrollbar_size_) redraw();
  scrollbar_size_ = newSize;
}

Fl_Scrollbar::Fl_Scrollbar(int X, int Y, int W, int H, const char *L = 0);

~Fl_Scrollbar();

int handle(int) FL_OVERRIDE;

int linesize() const { return linesize_; }

void linesize(int i) { linesize_ = i; }
```
This library is free software. Distribution and use rights are outlined in the file "COPYING" which should have been included with this file. If this file is missing or damaged, see the license at: https://www.fltk.org/COPYING.php
Please see the following page on how to report bugs and issues: https://www.fltk.org/bugs.php

```c
/*
file Fl_Secret_Input widget . */

#ifndef Fl_Secret_Input_H
#define Fl_Secret_Input_H

#include "Fl_Input.H"

class FL_EXPORT Fl_Secret_Input : public Fl_Input {
public:
  Fl_Secret_Input(int X,int Y,int W,int H,const char *l = 0);
  int handle(int) FL_OVERRIDE;
};
#endif
```

This library is free software. Distribution and use rights are outlined in the file "COPYING" which should have been included with this file. If this file is missing or damaged, see the license at: https://www.fltk.org/COPYING.php
Please see the following page on how to report bugs and issues: https://www.fltk.org/bugs.php

```c
/*
file Fl_Select_Browser widget . */

#ifndef Fl_Select_Browser_H
#define Fl_Select_Browser_H

#include "Fl_Browser.H"

class FL_EXPORT Fl_Select_Browser : public Fl_Browser {
public:
  Fl_Select_Browser(int X,int Y,int W,int H,const char *L=0);
};
#endif
```

This library is free software. Distribution and use rights are outlined in the file "COPYING" which should have been included with this file. If this file is missing or damaged, see the license at: https://www.fltk.org/COPYING.php
Please see the following page on how to report bugs and issues: https://www.fltk.org/bugs.php

```c
/*
file Fl_Shared_Image widget . */

#include "Fl_Image.H"

Classes

- class Fl_Shared_Image
  
  This class supports caching, loading, and drawing of image files.

Typedefs

- typedef Fl_Image *(* Fl_Shared_Handler) (const char *name, uchar *header, int headerlen)
Test function (typedef) for adding new shared image formats.

Functions

- void fl_register_images ()
  Register the known image formats.

13.129.1 Detailed Description

Fl_Shared_Image class.

13.129.2 Typedef Documentation

13.129.2.1 Fl_Shared_Handler

typedef Fl_Image *( Fl_Shared_Handler) (const char *name, uchar *header, int headerlen)

Test function (typedef) for adding new shared image formats.

This defines the function type you can use to add a handler for unknown image formats that can be opened and
loaded as an Fl_Shared_Image.

fl_register_images() adds all image formats known to FLTK. Call Fl_Shared_Image::add_handler() to add your own
check function to the list of known image formats.

Your function will be passed the filename (name), some header bytes already read from the image file and the
size headerlen of the data read. The max value of size is implementation dependent. If your handler function
needs to check more bytes you must open the image file yourself.

The provided buffer header must not be overwritten.

If your handler function can identify the file type you must open the file and return a valid Fl_Image or derived type,
otherwise you must return NULL. Example:

```c
static Fl_Image *check_my_image(const char *name,
uchar *header,
int headerlen) {
  // (test image type using header and headerlen)
  if (known) {
    // (load image data from file \p name)
    return new Fl_RGB_Image(data, ...);
  } else
    return 0;
}
```

// add your handler:
Fl_Shared_Image::add_handler(check_my_image);

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>filename to be checked and opened if applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>header</td>
<td>portion of the file that has already been read</td>
</tr>
<tr>
<td>in</td>
<td>headerlen</td>
<td>length of provided header data</td>
</tr>
</tbody>
</table>

Returns

valid Fl_Image or NULL.

See also

Fl_Shared_Image::add_handler()
**13.130 Fl_Shared_Image.H**

Go to the documentation of this file.

```cpp
#ifndef Fl_Shared_Image_H
#define Fl_Shared_Image_H

#include "Fl_Image.H"

#undef SHIM_DEBUG

typedef Fl_Image *(*Fl_Shared_Handler)(const char *name, uchar *header, int headerlen);

class FL_EXPORT Fl_Shared_Image : public Fl_Image {

protected:
    static Fl_Shared_Image **images_; // Shared images
    static int num_images_; // Number of shared images
    static int alloc_images_; // Allocated shared images
    static Fl_Shared_Handler *handlers_; // Additional format handlers
    static int num_handlers_; // Number of format handlers
    static int alloc_handlers_; // Allocated format handlers

    const char *name_; // Name of image file
    int original_; // Original image?
    int refcount_; // Number of times this image has been used
    Fl_Image *image_; // The image that is shared
    int alloc_image_; // Was the image allocated?

    static int compare(Fl_Shared_Image **i0, Fl_Shared_Image **i1);

public:
    #ifdef SHIM_DEBUG
    static void print_pool();
    #endif

    const char *name() { return name_; }
    int refcount() { return refcount_; }
    int original() { return original_; }
    Fl_Shared_Image *as_shared_image() FL_OVERRIDE { return this; }
    Fl_Image *copy(int W, int H) const FL_OVERRIDE;
    Fl_Image *copy() const;
    Fl_Image *copy();
    void color_average(Fl_Color c, float i) FL_OVERRIDE;
    void desaturate() FL_OVERRIDE;

    static void print_pool();

public:
    #ifdef SHIM_DEBUG
    static void print_pool();
    #endif

    static Fl_Shared_Image *copy_(int W, int H) const;
    Fl_Shared_Image();
    Fl_Shared_Image(const char *n, Fl_Image *img = 0);
    virtual ~Fl_Shared_Image();
    void add();
    void update();
    Fl_Shared_Image *copy_(int W, int H) const;

    Fl_Shared_Image *copy(int W, int H) const FL_OVERRIDE;
    Fl_Shared_Image *copy() const;
    Fl_Shared_Image *copy();
    void color_average(Fl_Color c, float i) FL_OVERRIDE;
    void desaturate() FL_OVERRIDE;
```

Generated by Doxygen
13.131 Fl_Shortcut_Button.H

00161 void draw(int X, int Y, int W, int H, int cx = 0, int cy = 0) FL_OVERRIDE;
00162 void draw(int X, int Y) { draw(X, Y, w(), h(), 0, 0); }
00163 void uncache() FL_OVERRIDE;
00164
00165 static Fl_Shared_Image *find(const char *name, int W = 0, int H = 0);
00166 static Fl_Shared_Image *get(const char *name, int W = 0, int H = 0);
00167 static Fl_Shared_Image *get(Fl_RGB_Byte *rgb, int owns_it = 1);
00168 static Fl_Shared_Image **images();
00169 static int num_images();
00170 static void add_handler(Fl_Shared_Handler f);
00171 static void remove_handler(Fl_Shared_Handler f);
00172
00194 const Fl_Image *image() const { return image_; }
00195
00196 }; // class Fl_Shared_Image
00197
00198 // The following function is provided in the fltk_images library and
00199 // registers all of the "extra" image file formats that are not part
00200 // of the core FLTK library...
00201 //
00202 //
00203 FL_EXPORT extern void fl_register_images();
00205
00206 #endif // Fl_Shared_Image_H

13.131 Fl_Shortcut_Button.H

00001 //
00002 // Shortcut Button header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2023 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file. If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 // https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 // https://www.fltk.org/bugs.php
00015 //
00016 //
00017 ifndef Fl_Shortcut_Button_H
00018 #define Fl_Shortcut_Button_H
00019
00020 #include <FL/Fl_Button.H>
00021
00022 class FLEXPORT Fl_Shortcut_Button : public Fl_Button {
00023 private:
00024 bool hot_, pre_hot_, default_set_, handle_default_button_;
00025 Fl_Shortcut pre_esc_;
00026 Fl_Shortcut default_shortcut_;
00027
00028 protected:
00029 Fl_Shortcut shortcut_value;
00030 void do_end_hot_callback();
00031 void handle(int) FL_OVERRIDE;
00032
00033 public:
00034 Fl_Shortcut_Button(int X, int Y, int W, int H, const char * l = 0);
00035 void value(Fl_Shortcut shortcut);
00036 Fl_Shortcut shortcut_value();
00037 #if 0
00038 // Default shortcut settings are disabled until successful review of the UI
00039 void default_value(Fl_Shortcut shortcut);
00040 Fl_Shortcut default_value();
00041 void default_clear();
00042 #endif
00043
00044 #endif // Fl_Shortcut_Button_H
00045

13.132 fl_show_colormap.H File Reference

The fl_show_colormap() function hides the implementation classes used to provide the popup window and color selection mechanism.
Functions

- Fl_Color fl_show_colormap (Fl_Color oldcol)
  
Pops up a window to let the user pick a colormap entry.

### 13.132.1 Detailed Description

The `fl_show_colormap()` function hides the implementation classes used to provide the popup window and color selection mechanism.

### 13.133 fl_show_colormap.H

**Go to the documentation of this file.**

```c
#ifndef fl_show_colormap_H
#define fl_show_colormap_H

/* doxygen comment here to avoid exposing ColorMenu in fl_show_colormap.cxx */

FL_EXPORT Fl_Color fl_show_colormap(Fl_Color oldcol);
#endif
```

### 13.134 fl_show_input.H

```c
// Standard input dialog header file for the Fast Light Tool Kit (FLTK).

#include "fl_ask.H"
```

### 13.135 Fl_Simple_Counter.H

```c
// Simple counter header file for the Fast Light Tool Kit (FLTK).

```

Generated by Doxygen
13.136 Fl_Single_Window.H

```cpp
#include "Fl_Window.H"

class FL_EXPORT Fl_Single_Window : public Fl_Window {
  public:
    Fl_Single_Window(int W, int H, const char *l=0);

    void show() FL_OVERRIDE;
    void show(int a, char **b) {Fl_Window::show(a,b);}

    Fl_Single_Window(int X, int Y, int W, int H, const char *l=0);

};
```

13.137 Fl_Slider.H

```cpp
#include "Fl_Valuator.H"

class FL_EXPORT Fl_Slider : public Fl_Valuator {
  public:
    Fl_Slider(int X, int Y, int W, int H, const char *l=0);

};
```
```
#define FL_VERT_SLIDER 0
#define FL_HOR_SLIDER 1
#define FL_VERTICAL_FILL_SLIDER 2
#define FL_HOR_FILL_SLIDER 3
#define FL_VERT_NICE_SLIDER 4
#define FL_HOR_NICE_SLIDER 5

class FL_EXPORT Fl_Slider : public Fl_Valuator {
    float slider_size_; 
    uchar slider_; 
    void _Fl_Slider(); 
    void draw_bg(int, int, int, int); 

protected: 
    // these allow subclasses to put the slider in a smaller area: 
    void draw(int, int, int, int); 
    int handle(int, int, int, int); 
    void draw(); 

public: 
    int handle(int); 
    Fl_Slider(int X,int Y,int W,int H, const char *L = 0); 
    Fl_Slider(uchar t,int X,int Y,int W,int H); 
    int scrollvalue(int pos,int size,int first,int total); 
    void bounds(double a, double b); 
    float slider_size(); 
    void slider_size(double v); 
    Fl_Boxtype slider() const {return (Fl_Boxtype)slider_;} 
    void slider(Fl_Boxtype c) {slider_ = c;} 
};
```

### 13.138 Fl_Spinner.H

```
#define Fl_Spinner_H

#include <FL/Enumerations.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Repeat_Button.H>

class FL_EXPORT Fl_Spinner : public Fl_Group {
    double value_; // Current value 
    double minimum_; // Minimum value 
    double maximum_; // Maximum value 
    double step_; // Amount to add/subtract for up/down 
    const char *format_; // Format string for input field 
    int wrap_; // wrap around at bounds (1/0) 

private: 
    static void sb_cb(Fl_Widget *w, Fl_Spinner *sb); // internal callback 
    void update(); // update input field 
};
```
protected:

// This class works like Fl_Input but ignores FL_Up and FL_Down key presses so they are handled by its parent, the Fl_Spinner widget.
// See STR #2989.
class FL_EXPORT Fl_Spinner_Input : public Fl_Input {
public:
  Fl_Spinner_Input(int X, int Y, int W, int H)
  : Fl_Input(X, Y, W, H) {}
  int handle(int event) FL_OVERRIDE; // implemented in src/Fl_Spinner.cxx;
}

Fl_Spinner_Input input_; // Input field for the value
Fl_Repeat_Button up_button_, // Up button
down_button_; // Down button

void draw() FL_OVERRIDE;

class Fl_Spinner(int X, int Y, int W, int H, const char *L = 0);

int handle(int event) FL_OVERRIDE;

void resize(int X, int Y, int W, int H) FL_OVERRIDE;

const char *format() const { return (format_); }

void format(const char *f) { format_ = f; update(); }

double maximum() const { return (maximum_); }

void maximum(double m) { maximum_ = m; }

double minimum() const { return (minimum_); }

void minimum(double m) { minimum_ = m; }

void range(double a, double b) { minimum_ = a; maximum_ = b; }

void step(double s);

double step() const { return (step_); }

void wrap(int set) { wrap_ = set ? 1 : 0; }

int wrap() const { return wrap_; }

Fl_Color textcolor() const { return (input_.textcolor()); }

void textcolor(Fl_Color c) { input_.textcolor(c); }

Fl_Font textfont() const { return (input_.textfont()); }

void textfont(Fl_Font f) { input_.textfont(f); }

Fl_Fontsize textsize() const { return (input_.textsize()); }

void type(uchar v);

uchar type() const { return (input_.type()); }

double value() const { return (value_); }

void value(double v) { value_ = v; update(); }

void color(Fl_Color c) { input_.color(c); }

Fl_Color color() const { return (input_.color()); }

void selection_color(Fl_Color val) { input_.selection_color(val); }

void maximum_size(int m) { if (m > 0) input_.maximum_size(m); }

int maximum() const { return (input_.maximum_size()); }

};
Public header for FLTK's platform-agnostic string handling.

#include "Fl_Export.H"

Functions

• char * fl_strdup (const char *s)
  
  Cross platform interface to POSIX function strdup().

13.139.1 Detailed Description

Public header for FLTK's platform-agnostic string handling.

13.140 fl_string_functions.h

Go to the documentation of this file.

13.141 Fl_SVG_File_Surface.H

Decleration of Fl_SVG_File_Surface in the Fast Light Tool Kit (FLTK).
```c
#include <FL/Fl_Widget_Surface.H>
#include <stdio.h>

class FL_EXPORT Fl_SVG_File_Surface : public Fl_Widget_Surface {
  int width_, height_;  
  int ( *closef_)(FILE*);
  public:
    Fl_SVG_File_Surface(int width, int height, FILE *svg, int ( *closef)(FILE*) = NULL);
    ~Fl_SVG_File_Surface();
    FILE *file();
    void origin(int x, int y) FL_OVERRIDE;
    void origin(int *x, int *y) FL_OVERRIDE;
    void translate(int x, int y) FL_OVERRIDE;
    void untranslate() FL_OVERRIDE;
    int printable_rect(int *w, int *h) FL_OVERRIDE;
    int close();
};
```

```c
#include <FL/Fl_Image.H>

struct NSVGimage;

class FL_EXPORT Fl_SVG_Image : public Fl_RGB_Image {
  private:
    typedef struct {
      NSVGimage * svg_image;
      int ref_count;
    } counted_NSVGimage;
    counted_NSVGimage * counted_svg_image_;  
    bool rasterized_;   
    int raster_w_, raster_h_;           
    bool to_desaturate_;     
    Fl_Color average_color_;   
    float average_weight_;   
    float svg_scaling_(int W, int H);   
    void rasterize_(int W, int H);  
    void cache_size_(int &width, int &height) FL_OVERRIDE;
    void init_(const char *name, const unsigned char *filedata, size_t length);
    Fl_SVG_Image(const Fl_SVG_Image *source);
  public:
    bool proportional;
    Fl_SVG_Image(const char *filename);
    Fl_SVG_Image(const char *sharedname, const char *svg_data);
    Fl_SVG_Image(const char *sharedname, const unsigned char *svg_data, size_t length);
    virtual ~Fl_SVG_Image();
    Fl_Image *copy(int W, int H) const FL_OVERRIDE;
    Fl_Image *copy() const {
      return Fl_Image::copy();
    }
    void resize(int width, int height);
    void desaturate() FL_OVERRIDE;
    void color_average(Fl_Color c, float i) FL_OVERRIDE;
    void draw(int X, int Y, int W, int H, int cx = 0, int cy = 0) FL_OVERRIDE;
    void draw(int X, int Y) { draw(X, Y, w(), h(), 0, 0); }
    Fl_SVG_Image *as_svg_image() FL_OVERRIDE { return this; }
    void normalize() FL_OVERRIDE;
};
```
13.143 Fl_Sys_Menu_Bar.H File Reference

Definition of class Fl_Sys_Menu_Bar.
#include <FL/Fl_Menu_Bar.H>

Classes

• class Fl_Sys_Menu_Bar
  A class to create and modify menus that appear on macOS in the menu bar at the top of the screen.

Variables

• Fl_Sys_Menu_Bar * fl_sys_menu_bar
  The system menu bar.

13.143.1 Detailed Description

Definition of class Fl_Sys_Menu_Bar.

13.144 Fl_Sys_Menu_Bar.H

Go to the documentation of this file.

```c
#ifndef Fl_Sys_Menu_Bar_H
#define Fl_Sys_Menu_Bar_H

#include <FL/Fl_Menu_Bar.H>

class Fl_Sys_Menu_Bar_Driver;

class FL_EXPORT Fl_Sys_Menu_Bar : public Fl_Menu_Bar {
  static Fl_Sys_Menu_Bar_Driver *driver();
  protected:
    void draw() FL_OVERRIDE;
  public:
    typedef enum {
      no_window_menu = 0,
      tabbing_mode_none,
      tabbing_mode_automatic,
      tabbing_mode_preferred
    } window_menu_style_enum;

    Fl_Sys_Menu_Bar(int x, int y, int w, int h, const char * l=0);
    virtual ~Fl_Sys_Menu_Bar();

    const Fl_Menu_Item * menu() const {return Fl_Menu_::menu();}
    void menu(const Fl_Menu_Item * m);

    int add(const char * label, int shortcut, Fl_Callback *, void * user_data=0, int flags=0);
    int add(const char * label, const char* shortcut, Fl_Callback * cb, void * user_data=0, int flags=0) {
      return add(label, fl_old_shortcut(shortcut), cb, user_data, flags);
    }

    int add(const char * str);

    int insert(int index, const char * label, int shortcut, Fl_Callback * cb, void * user_data=0, int flags=0);
    int insert(int index, const char * label, const char* shortcut, Fl_Callback * cb, void * user_data=0, int flags=0) {
      return insert(index, label, fl_old_shortcut(shortcut), cb, user_data, flags);
    }

    void remove(int n);

    void replace(int index, const char *name);
```
void clear();
int clear_submenu(int index);
void mode (int i, int fi);
int mode(int i) const { return Fl_Menu_::mode(i); };
void setonly (Fl_Menu_Item *item);
static void about(Fl_Callback *cb, void *data);
static window_menu_style_enum window_menu_style();
static void window_menu_style(window_menu_style_enum style);
static void create_window_menu();

extern Fl_Sys_Menu_Bar *fl_sys_menu_bar;

#ifndef _FL_TABLE_H
#define _FL_TABLE_H

#include <FL/Fl_Group.H>
#include <FL/Fl_Scroll.H>

// EXPERIMENTAL
// We use either std::vector or the private class Fl_Int_Vector
// depending on the build option OPTION_USE_STD or --enable-use_std.
// This option allows to use std::string and std::vector in FLTK 1.4.x
#if (FLTK_USE_STD)
#include <vector>
typedef std::vector<int> Fl_Int_Vector;
#else
class Fl_Int_Vector; // private class declared in src/Fl_Int_Vector.H
#endif

class FL_EXPORT Fl_Table : public Fl_Group {
enum TableContext {
CONTEXT_NONE = 0,
CONTEXT_STARTPAGE = 0x01,
CONTEXT_ENDPAGE = 0x02,
CONTEXT_ROW_HEADER = 0x04,
CONTEXT_COL_HEADER = 0x08,
CONTEXT_CELL = 0x10,
CONTEXT_TABLE = 0x20,
CONTEXT_RC_RESIZE = 0x40
};

private:
int _rows, _cols; // total rows/cols
int _row_header_w; // width of row header
int _col_header_h; // height of column header
int _row_position; // last row_position set (not necessarily == toprow!)
int _col_position; // last col_position set (not necessarily == leftcol!)
char _row_header; // row header enabled?
char _col_header; // col header enabled?
char _row_resize; // row resizing enabled?
char _col_resize; // col resizing enabled?
int _row_resize_min; // row minimum resizing height (default=1)
int _col_resize_min; // col minimum resizing width (default=1)

// OPTIMIZATION: partial row/column redraw variables
int _redraw_toprow;
int _redraw_botrow;
int _redraw_leftcol;
int _redraw_rightcol;
Fl_Color _row_header_color;
Fl_Color _col_header_color;

int _auto_drag;
int _selecting;
int _scrollbar_size;
enum {
  TABCELLNAV = 1
};
unsigned int flags_;

Fl_Int_Vector * _colwidths; // column widths in pixels
Fl_Int_Vector * _rowheights; // row heights in pixels

// number of columns and rows == size of corresponding vectors
int col_size(); // size of the column widths vector
int row_size(); // size of the row heights vector

Fl_Cursor _last_cursor; // last mouse cursor before changed to 'resize' cursor

// EVENT CALLBACK DATA
TableContext _callback_context; // event context
int _callback_row, _callback_col; // event row/col

// handle() state variables.
// Put here instead of local statics in handle(), so more
// than one Fl_Table can exist without crosstalk between them.
int _resizing_col; // column being dragged
int _resizing_row; // row being dragged
int _dragging_x; // starting x position for horiz drag
int _dragging_y; // starting y position for vert drag
int _last_row; // last row we FL_PUSH'ed

// Redraw single cell
void _redraw_cell(TableContext context, int R, int C);
void _start_auto_drag();
void _stop_auto_drag();
static void _auto_drag_cb2(void *d);

protected:
enum ResizeFlag {
  RESIZE_NONE = 0,
  RESIZE_COL_LEFT = 1,
  RESIZE_COL_RIGHT = 2,
  RESIZE_ROW_ABOVE = 3,
  RESIZE_ROW_BELOW = 4
};

int table_w;
int table_h;

// selection
int current_row;
int current_col;
int select_row;
int select_col;

// OPTIMIZATION: Precomputed scroll positions for the toprow/leftcol
int toprow_scrollpos;
int leftcol_scrollpos;

// Data table’s inner dimension
int tix;
int tiy;

// Data table’s outer dimension
int tow;
int toy;

// Table widget’s inner dimension
int wix;
int wiy;
virtual void draw_cell(TableContext context, int R=0, int C=0, int X=0, int Y=0, int W=0, int H=0); // overridden by deriving class

todo: add topline(), middleline(), bottomline()
inline int cols() {
    return(_cols);
}

inline void visible_cells(int & r1, int & r2, int & c1, int & c2) {
    r1 = toprow;
    r2 = botrow;
    c1 = leftcol;
    c2 = rightcol;
}

int is_interactive_resize() {
    return(_resizing_row != -1 || _resizing_col != -1);
}

inline int row_resize() {
    return(_row_resize);
}

void row_resize(int flag) { // enable row resizing
    _row_resize = flag;
}

inline int col_resize() {
    return(_col_resize);
}

void col_resize(int flag) { // enable col resizing
    _col_resize = flag;
}

inline int col_resize_min() { // column minimum resizing width
    return(_col_resize_min);
}

void col_resize_min(int val) {
    _col_resize_min = ( val < 1 ) ? 1 : val;
}

inline int row_resize_min() { // column minimum resizing width
    return(_row_resize_min);
}

void row_resize_min(int val) {
    _row_resize_min = ( val < 1 ) ? 1 : val;
}

inline int row_header() { // set/get row header enable flag
    return(_row_header);
}

void row_header(int flag) {
    _row_header = flag;
    table_resized();
    redraw();
}

inline int col_header() { // set/get col header enable flag
    return(_col_header);
}

void col_header(int flag) {
    _col_header = flag;
    table_resized();
    redraw();
}

inline void col_header_height(int height) { // set/get col header height
    _col_header_h = height;
    table_resized();
    redraw();
}

inline int col_header_height() {
    return(_col_header_h);
}

inline void row_header_width(int width) { // set/get row header width
    _row_header_w = width;
    table_resized();
    redraw();
}

inline int row_header_width() {
    return(_row_header_w);
}
inline void row_header_color(Fl_Color val) { // set/get row header color
  _row_header_color = val;
  redraw();
}
inline Fl_Color row_header_color() {
  return(_row_header_color);
}
inline void col_header_color(Fl_Color val) { // set/get col header color
  _col_header_color = val;
  redraw();
}
inline Fl_Color col_header_color() {
  return(_col_header_color);
}

void row_height(int row, int height); // set row height in pixels
int row_height(int row);

void col_width(int col, int width); // set a column's width in pixels
int col_width(int col);

void row_height_all(int height) { // set all row/col heights
  for ( int r=0; r<rows(); r++ ) {
    row_height(r, height);
  }
}

void col_width_all(int width) {
  for ( int c=0; c<cols(); c++ ) {
    col_width(c, width);
  }
}

inline void top_row(int row) { // set/get top row (deprecated)
  row_position(row);
}
inline int top_row() {
  return(row_position());
}
int is_selected(int r, int c); // selected cell
void get_selection(int &row_top, int &col_left, int &row_bot, int &col_right);
void set_selection(int row_top, int col_left, int row_bot, int col_right);

void move_cursor(int R, int C, int shiftselect);
void resize(int X, int Y, int W, int H) FL_OVERRIDE; // fltk resize() FL_OVERRIDE

// Child group management

void init_sizes() {
  table->init_sizes();
  table->redraw();
}
void add(Fl_Widget& wgt) {
  table->add(wgt);
  if ( table->children() > 2 ) {
    table->show();
  }
}

// This crashes sortapp() during init.
void box(Fl_Boxtype val) {
  if ( table ) {
    resize(x(), y(), w(), h());
  }
}

// Child group management

void init_sizes() {
  table->init_sizes();
  table->redraw();
}
void add(Fl_Widget& wgt) {
  table->add(wgt);
  if ( table->children() > 2 ) {
    table->show();
  }

Generated by Doxygen
void add(Fl_Widget * wgt) {
  add(*wgt);
}

void insert(Fl_Widget& wgt, int n) {
  table->insert(wgt,n);
}

void insert(Fl_Widget& wgt, Fl_Widget* w2) {
  table->insert(wgt, w2);
}

void remove(Fl_Widget& wgt) {
  table->remove(wgt);
}

void begin() {
  table->begin();
}

void end() {
  table->end();
  // HACK: Avoid showing Fl_Scroll; seems to erase screen
  // causing unnecessary flicker, even if its box() is FL_NO_BOX.
  //
  if (table->children() > 2) {
    table->show();
    } else {
    table->hide();
  }
  Fl_Group::current(Fl_Group::parent());
}

Fl_Widget *const* array() {
  return(table->array());
}

Fl_Widget * child(int n) const {
  return(table->child(n));
}

int children() const {
  return(table->children()-2);  // -2: skip Fl_Scroll's h/v scrollbar widgets
}

int find(const Fl_Widget* wgt) const {
  return(table->find(wgt));
}

int find(const Fl_Widget& wgt) const {
  return(table->find(wgt));
}

// CALLBACKS

int callback_row() {
  return(_callback_row);
}

int callback_col() {
  return(_callback_col);
}

TableContext callback_context() {
  return(_callback_context);
}

void do_callback(TableContext context, int row, int col) {
  _callback_context = context;
  _callback_row = row;
  _callback_col = col;
  Fl_Widget::do_callback();
}

ifdef FL_DOXYGEN
void when(Fl_When flags);
endif

Generated by Doxygen
# Fl_Table_Row.H

```
0001 //
0002 ifndef _FL_TABLE_ROW_H
0003 #define _FL_TABLE_ROW_H
0004 #include <FL/Fl_Table.H>
0005 class FL_EXPORT Fl_Table_Row : public Fl_Table {
0006    enum TableRowSelectMode {
0007        SELECT_NONE, // no selection allowed
0008        SELECT_SINGLE, // single row selection
0009        SELECT_MULTI // multiple row selection (default)
0010    };
0011 private:
0012    CharVector* char_arr;
0013    int _size;
0014    void init() {
0015        arr = 0;
0016        _size = 0;
0017    }
0018    void copy(CharVector* newarr, int newsize);
0019 public:
0020    Fl_Table_Row(); // CTOR
0021    Fl_Table_Row& operator=(CharVector* o); // ASSIGN
0022    Fl_Table_Row* operator=(CharVector* o); // CTOR
0023    void copy(CharVector* o, int x); // CTOR
0024    void operator[](int x) const {
```
return(arr[x]);
}

int size() {
  return(_size);
}
void size(int count); // set number of rows
char pop_back() {
  char tmp = arr[_size-1];
  _size--;
  return(tmp);
}
void push_back(char val) {
  int x = _size;
  size(_size+1);
  arr[x] = val;
}
char back() {
  return(arr[_size-1]);
}
};

CharVector _rowselect; // selection flag for each row

int _dragging_select; // dragging out a selection?

int _last_row;
int _last_y; // last event's Y position
int _last_push_x; // last PUSH event's X position
int _last_push_y; // last PUSH event's Y position
TableRowSelectMode _selectmode;

protected:
int handle(int event) FL_OVERRIDE;
int find_cell(TableContext context, // find cell's x/y/w/h given r/c
  int R, int C, int &X, int &Y, int &W, int &H) {
  return(Fl_Table::find_cell(context, R, C, X, Y, W, H));
}

public:
Fl_Table_Row(int X, int Y, int W, int H, const char *l=0) : Fl_Table(X,Y,W,H,l) {
  _dragging_select = 0;
  _last_row = -1;
  _last_y = -1;
  _last_push_x = -1;
  _last_push_y = -1;
  _selectmode = SELECT_MULTI;
}

int row_selected(int row); // is row selected? (0=no, 1=yes, -1=range err)
int select_row(int row, int flag=1); // select state for row: flag:0=off, 1=on, 2=toggle
  // returns: 0=no change, 1=changed, -1=range err
void select_all_rows(int flag=1); // all rows to a known state
void clear() FL_OVERRIDE {
  rows[0]; // implies clearing selection
cols[0];
  Fl_Table::clear(); // clear the table
}

};

#if 0
#endif
13.147 Fl_Tabs.H

```
/* file
Fl_Tabs widget. */

#ifndef Fl_Tabs_H
#define Fl_Tabs_H

#include "Fl_Group.H"

struct Fl_Menu_Item;

class FL_EXPORT Fl_Tabs : public Fl_Group {

Fl_Widget *push_;

protected:

int overflow_type;
int tab_offset;
int *tab_pos;
int *tab_width;
int *tab_flags;
int tab_count;
int has_overflow_menu;
Fl_Align tab_align_;

public:

Fl_Tabs(int X, int Y, int W, int H, const char *L = 0);
virtual ~Fl_Tabs();

int handle(int) FL_OVERRIDE;
Fl_Widget *value();
int value(Fl_Widget *);

Fl_Widget *push() const { return push_; }
int push(Fl_Widget *);

virtual Fl_Widget *which(int event_x, int event_y);
void client_area(int &rx, int &ry, int &rw, int &rh, int tabh=0);

void tab_align(Fl_Align a) { tab_align_ = a; }
enum {
    OVERFLOW_COMPRESS = 0,
    OVERFLOW_CLIP,
    OVERFLOW_PULLDOWN,
    OVERFLOW_DRAG
};

```

void handle_overflow(int ov);

#endif

13.148 Fl_Terminal.H File Reference

Fl_Terminal widget.

#include <FL/Fl.H>
#include <FL/Fl_Terminal.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_History.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Scrollbar.H>
#include <FL/Fl_Rect.H>
#include <stdarg.h>

Classes

- class Fl_Terminal::CharStyle
- class Fl_Terminal::Cursor
- class Fl_Terminal::EscapeSeq
- class Fl_Terminal

Terminal widget supporting Unicode/utf-8, ANSI/xterm escape codes with full RGB color control.

- class Fl_Terminal::Margin
- class Fl_Terminal::PartialUtf8Buf
- class Fl_Terminal::RingBuffer
- class Fl_Terminal::Selection
- class Fl_Terminal::Utf8Char

13.148.1 Detailed Description

Fl_Terminal widget.

13.149 Fl_Terminal.H

Go to the documentation of this file.

#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Scrollbar.H>
#include <FL/Fl_Rect.H>

#include <stdarg.h> // va_list (MinGW)

class FL_EXPORT Fl_Terminal : public Fl_Group {
```c
public:
  enum RedrawStyle {
    NO_REDRAW=0,
    RATE_LIMITED,
    PER_WRITE
  };

enum Attrib {
  NORMAL = 0x00,
  BOLD = 0x01,
  DIM = 0x02,
  ITALIC = 0x04,
  UNDERLINE = 0x08,
  _RESERVED_1 = 0x10,
  INVERSE = 0x20,
  _RESERVED_2 = 0x40,
  STRIKEOUT = 0x80
};

enum CharFlags {
  FG_XTERM = 0x01,
  BG_XTERM = 0x02,
  ESC = 0x03,
  RESV_A = 0x08,
  RESV_B = 0x10,
  RESV_C = 0x20,
  RESV_D = 0x40,
  RESV_E = 0x80,
  _RESERVED_2 = 0x40,
  STRIKEOUT = 0x80
};

enum OutFlags {
  OFF = 0x00,
  CR_TO_LF = 0x01,
  LF_TO_CR = 0x02,
  LF_TO_CRLF = 0x04
};

protected:
  // Margin Class ///////////////////////////////////////////
  // Class to manage the terminal’s margins
  class FL_EXPORT Margin {
    int left_, right_, top_, bottom_;
  public:
    Margin(void) { left_ = right_ = top_ = bottom_ = 3; }
    int left(void) const { return left_; }
    int right(void) const { return right_; }
    int top(void) const { return top_; }
    int bottom(void) const { return bottom_; }
    void left(int val) { left_ = val; }
    void right(int val) { right_ = val; }
    void top(int val) { top_ = val; }
    void bottom(int val) { bottom_ = val; }
  };

  // CharStyle Class ///////////////////////////////////////////
  // Class to manage the terminal’s character style
  // This includes the font, color, and some cached internal
  // info for optimized drawing speed.
  class FL_EXPORT CharStyle {
    uchar attrib_; // bold, underline..
    uchar charflags_; // CharFlags (xterm color management)
    Fl_Color fgcolor_; // foreground color for text
    Fl_Color bgcolor_; // background color for text
    Fl_Color defaultfgcolor_; // default fg color used by ESC[0m
    Fl_Color defaultbgcolor_; // default bg color used by ESC[0m
    Fl_Font fontface_; // font face
    Fl_Fontsize fontsize_; // font size
    int fontheight_; // font height (in pixels)
    int fontdescent_; // font descent (pixels below font baseline)
    int charwidth_; // width of a fixed width ASCII character
  public:
    CharStyle(bool fontsize_defer);
    uchar attrib(void) const { return attrib_; }
    uchar charflags(void) const { return charflags_; }
    Fl_Color fgcolor(void) const;
    Fl_Color bgcolor(void) const;
    Fl_Color defaultfgcolor(void) const;
    Fl_Color defaultbgcolor(void) const;
    Fl_Font fontface(void) const { return fontface_; }
    Fl_Fontsize fontsize(void) const { return fontsize_; }
    int fontheight(void) const { return fontheight_; }
    int fontdescent(void) const { return fontdescent_; }
  };
```
int charwidth(void) const { return charwidth_; }

uchar colorbits_only(uchar inflags) const;

uchar attrib(uchar val) { attrib_ = val; }

uchar charflags(uchar val) { charflags_ = val; }

uchar clr_charflag(uchar val) { charflags_ &= ~val; }

uchar fgcolor_uchar(uchar val);

uchar bgcolor_uchar(uchar val);

void attrib(uchar val) { attrib_ = val; }

void charflags(uchar val) { charflags_ = val; }

void set_charflag(uchar val) { charflags_ |= val; }

void clr_charflag(uchar val) { charflags_ &= ~val; }

void fgcolor(int r, int g, int b) { fgcolor_ = (r << 24) | (g << 16) | (b << 8); clr_charflag(FG_XTERM); }

void bgcolor(int r, int g, int b) { bgcolor_ = (r << 24) | (g << 16) | (b << 8); clr_charflag(BG_XTERM); }

void fgcolor(Fl_Color val) { fgcolor_ = val; clr_charflag(FG_XTERM); }

void bgcolor(Fl_Color val) { bgcolor_ = val; clr_charflag(BG_XTERM); }

void defaultfgcolor(Fl_Color val) { defaultfgcolor_ = val; }

void defaultbgcolor(Fl_Color val) { defaultbgcolor_ = val; }

void fontface(Fl_Font val) { fontface_ = val; update(); }

void fontsize(Fl_Fontsize val) { fontsize_ = val; update(); }

void update(void);

void update_fake(void);

// SGR MODES: Set Graphics Rendition

int onoff(bool flag, Attrib a) { return (flag ? (attrib_ | a) : (attrib_ & ~a)); }

void sgr_bold(bool val) { attrib_ = onoff(val, Fl_Terminal::BOLD); }

void sgr_dim(bool val) { attrib_ = onoff(val, Fl_Terminal::DIM); }

void sgr_italic(bool val) { attrib_ = onoff(val, Fl_Terminal::ITALIC); }

void sgr_underline(bool val) { attrib_ = onoff(val, Fl_Terminal::UNDERLINE); }

void sgr_dbl_under(bool val) { attrib_ = onoff(val, Fl_Terminal::UNDERLINE); }

// NOT IMPLEMENTED

void sgr_blink(bool val) { (void)val; /* NOT IMPLEMENTED */ }

void sgr_inverse(bool val) { attrib_ = onoff(val, Fl_Terminal::INVERSE); }

void sgr_strike(bool val) { attrib_ = onoff(val, Fl_Terminal::STRIKEOUT); }

protected:

// Cursor Class /////////////////////////////////////////////////////////////////

// Class to manage the terminal’s cursor position, color, etc.

// Class FL_EXPORT Cursor

int col(void) const { return col_; }

int row(void) const { return row_; }

int h(void) const { return h_; }

Fl_Color fgcolor(void) const { return fgcolor_; }

Fl_Color bgcolor(void) const { return bgcolor_; }

// Utf8Char Class /////////////////////////////////////////////////////////////////

// Class to manage the terminal’s individual UTF-8 characters.

// Includes fg/bg color, attributes (BOLD, UNDERLINE...)

Generated by Doxygen
uchar len_;  // length of bytes in text_[ ] buffer; 1 for ASCII, >1 for UTF-8
uchar attrib_;  // attribute bits for this char (bold, underline,..)
uchar charflags_;  // CharFlags (xterm colors management)
Fl_Color fgcolor_;  // fltk fg color (supports 8color or 24bit color set
w/ESC[37;<r>;<g>;<b>m)
Fl_Color bgcolor_;  // fltk bg color (supports 8color or 24bit color set
w/ESC[47;<r>;<g>;<b>m)

// Private methods
void text_utf8_(const char *text, int len);

// Public:
public:

// Public methods
Utf8Char(void);  // ctor
Utf8Char(const Utf8Char& o);  // copy ctor
Utf8Char(Utf8Char&);  // dtor
Utf8Char& operator=(const Utf8Char& o);  // assignment

inline int max_utf8() const { return max_utf8_; }
void text_utf8(const char *text, int len, const CharStyle& style);
void text_ascii(char c, const CharStyle& style);
void fl_font_set(const CharStyle& style) const;

// Return the UTF-8 text string for this character.
// Use length() to get number of bytes in string, which will be 1 for ASCII chars.
const char * text_utf8(void) const { return text_; }

// Return the attribute for this char
uchar attrib(void) const { return attrib_; }
uchar charflags(void) const { return charflags_; }
Fl_Color fgcolor(void) const;
Fl_Color bgcolor(void) const;

// Return the length of this character in bytes (UTF-8 can be multibyte..)
int length(void) const { return int(len_); }
double pwidth(void) const;
int pwidth_int(void) const;

// Clear the character to a 'space'
void clear(const CharStyle& style) { text_ascii(' ', style); }

bool is_char(char c) const { return *text_ == c; }
void show_char(void) const { ::printf("%. *s", len_, text_); }
void show_char_info(void) const { ::fprintf(stderr, "UTF-8('%. *s'), len=%d)\n", len_, text_, len_); }


// Return the attribute for this char
uchar attrib(void) const { return attrib_; }
uchar charflags(void) const { return charflags_; }
Fl_Color fgcolor(void) const;
Fl_Color bgcolor(void) const;

// Return the attribute for this char
uchar attrib(void) const { return attrib_; }
uchar charflags(void) const { return charflags_; }
Fl_Color fgcolor(void) const;
Fl_Color bgcolor(void) const;

// Return the length of this character in bytes (UTF-8 can be multibyte..)
int length(void) const { return int(len_); }
double pwidth(void) const;
int pwidth_int(void) const;

// Clear the character to a 'space'
void clear(const CharStyle& style) { text_ascii(' ', style); }

bool is_char(char c) const { return *text_ == c; }
void show_char(void) const { ::printf("%. *s", len_, text_); }
void show_char_info(void) const { ::fprintf(stderr, "UTF-8('%. *s'), len=%d)\n", len_, text_, len_); }


// RingBuffer Class //////////////////////////////////////////////////////////////////

// Manages ring with indexed row/col and "history" vs. "display" concepts.

// _____________
// | | <- hist_srow() <- ring_srow()
// | H i s t |
// | |

Generated by Doxygen
inline int ring_rows(void) const { return ring_rows_; }
inline int ring_cols(void) const { return ring_cols_; }
inline int ring_srow(void) const { return(0); }
inline int ring_erow(void) const { return(ring_rows_ - 1); }
inline int hist_rows(void) const { return hist_rows_; }
inline int hist_cols(void) const { return ring_cols_; }
inline int hist_srow(void) const { return((offset_ + 0) % ring_rows_); }
inline int hist_erow(void) const { return((offset_ + hist_rows_ - 1) % ring_rows_); }
inline int disp_rows(void) const { return disp_rows_; }
inline int disp_cols(void) const { return ring_cols_; }
inline int disp_srow(void) const { return((offset_ + hist_rows_) % ring_rows_); }
inline int disp_erow(void) const { return((offset_ + hist_rows_ + disp_rows_ - 1) % ring_rows_); }
inline int offset(void) const { return offset_; }
void offset_adjust(int rows);
void hist_rows(int val) { hist_rows_ = val; }
void disp_rows(int val) { disp_rows_ = val; }

// History use
inline int hist_use(void) const { return hist_use_; }
inline void hist_use(int val) { hist_use_ = val; }
inline int hist_use_srow(void) const { return((offset_ + hist_rows_ - hist_use_) % ring_rows_); }

inline Utf8Char *ring_chars(void) { return ring_chars_; } // access ring buffer directly
inline Utf8Char *ring_chars(void) const { return ring_chars_; } // access ring buffer directly

bool is_hist_ring_row(int grow) const;
bool is_disp_ring_row(int grow) const;
//DEBUG void show_ring_info(void) const;

void move_disp_row(int src_row, int dst_row); // Non-const versions of the above methods
void clear_disp_rows(int sdrow, int edrow, const CharStyle& style);
void scroll(int rows, const CharStyle& style);

const Utf8Char * u8c_ring_row(int row) const;
const Utf8Char * u8c_hist_row(int hrow) const;
const Utf8Char * u8c_hist_use_row(int hurow) const;
const Utf8Char * u8c_disp_row(int drow) const;
// Non-const versions of the above methods
Utf8Char * u8c_ring_row(int row);
Utf8Char * u8c_hist_row(int hrow);
Utf8Char * u8c_hist_use_row(int hurow);
Utf8Char * u8c_disp_row(int drow);

void create(int drows, int dcols, int hrows);
void resize(int drows, int dcols, int hrows, const CharStyle& style);
void change_disp_rows(int drows, const CharStyle& style);
void change_disp_cols(int dcols, const CharStyle& style);

// Selection Class /////////////////////////////////////////////////////////
//
// Class to manage mouse selection
//
class FL_EXPORT Selection {
int srow_, scol_, erow_, ecol_; // selection start/end. NOTE: start *might* be > end
int push_row_, push_col_; // global row/col for last FL_PUSH

FL_Color selectionbgcolor_; // selection background color
FL_Color selectionfgcolor_; // selection foreground color
int state_; // 0=none, 1=started, 2=extended, 3=done

public:
Selection(void);
int srow(void) const { return srow_; }
int scol(void) const { return scol_; }
int erow(void) const { return erow_; }
int ecol(void) const { return ecol_; }
void push_clear() { push_row_ = push_col_ = -1; }
void push_rowcol(int row,int col) { push_row_ = row; push_col_ = col; }
void start_push() { start(push_row_, push_col_); }
bool dragged_off(int row,int col) { return (push_row_ != row) || (push_col_ != col); }

void selectionfgcolor(FL_Color val) { selectionfgcolor_ = val; }
void selectionfgcolor(FL_Color val) const { return selectionfgcolor_; }
void selectionbgcolor(FL_Color val) { selectionbgcolor_ = val; }
void selectionbgcolor(FL_Color val) const { return selectionbgcolor_; }
bool is_selection(void) const { return is_selection_; }

bool get_selection(int &srow, int &scol, int &erow, int &ecol) const; // guarantees return (start < end)

bool start(int row, int col);

bool extend(int row, int col);

void end(void);

void select(int srow, int scol, int erow, int ecol);

bool clear(void);

int state(void) const { return state_; }

void scroll(int nrows);

};

// EscapeSeq Class ///////////////////////////////////////////////////////////////////////////////////

// Class to handle parsing ESC sequences

// Holds all state information for parsing esc sequences,
// so sequences can span multiple block read(2) operations, etc.

// Handling of parsed sequences is NOT handled in this class,
// just the parsing of the sequences and managing generic integers.

class FL_EXPORT EscapeSeq {

public:

// EscapeSeq Constants

// Maximums

static const int maxbuff = 80; // character buffer

static const int maxvals = 20; // integer value buffer

// Return codes

static const int success = 0; // operation succeeded

static const int fail = -1; // operation failed

static const int completed = 1; // multi-step operation completed successfully

private:

char esc_mode_; // escape parsing mode state

csi_; // This is an ESC[.. sequence (Ctrl Seq Introducer)

char *buffp_; // parsing ptr into buff[]

char *buffendp_; // end of buff[] (ptr to last valid buff char)

char *valbuffp_; // pointer to first char in buff of integer being parsed

int vals_[maxvals]; // value array for parsing #'s in ESC[[#;#;#..

int vali_; // parsing index into vals_), 0 if none

int save_row_, save_col_; // used by ESC[s/u for save/restore

int append_buff(char c);

int append_val(void);

public:

EscapeSeq(void);

void reset(void);

char esc_mode(void) const;

void esc_mode(char val);

int total_vals(void) const;

int val(int i) const;

int defvalmax(int dval, int max) const;

bool parse_in_progress(void) const;

bool is_csi(void) const;

int parse(char c);

void save_cursor(int row, int col);

void restore_cursor(int &row, int &col);

};

// Partial UTF-8 Buffer Class //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Class to manage buffering partial UTF-8 characters between write calls.

// Partial Utf8Buf Class /////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

class FL_EXPORT PartialUtf8Buf {

char buf_[10]; // buffer partial UTF-8 encoded char

int buflen_; // length of buffered UTF-8 encoded char

int clen_; // final byte length of a UTF-8 char

void clear(void) { buflen_ = clen_ = 0; } // clear the buffer

void PartialUtf8Buf(void) { clear(); } // Ctor

// Is byte 'c' in the middle of a UTF-8 encoded byte sequence?

bool is_continuation(char c) {

// ASCII: 0xxxxxxx

// UTF8(2): 110xxx 10xxxxx

// UTF8(3): 1110xx 10xxxxx 10xxxxx

// UTF8(6): 11110x 10xxxxx 10xxxxx 10xxxxx 10xxxxx

// UTF8(8): 111110x 10xxxxx 10xxxxx 10xxxxx 10xxxxx 10xxxxx

// Start byte Continuation bytes

return ((c & 0xc0) == 0x80);

}

// Access buffer

const char* buf(void) const { return buf_; }

Generated by Doxygen
int buflen(void) const { return buflen_; }

bool append(const char * p, int len) {
    if (len <= 0) return true; // ignore silly requests: say we did but dont
    if (buflen_ + len >= (int)sizeof(buf_)) // overrun check
        { clear(); return false; } // clear self, return false
    if (!buflen_) clen_ = fl_utf8len(*p); // first byte? save char len for later
    while (len>0) { buf_[buflen_++] = *p++; len--;} // append byte to buffer
    return true;
}

bool is_complete(void) const { return (buflen_ && (buflen_ == clen_)); }

public:

Fl_Scrollbar *scrollbar; // vertical scrollbar (value: rows above disp_chars[])

private:

bool fontsize_defer_; // flag defers font calcs until first draw() (issue 837)
int scrollbar_size_; // local preference for scrollbar size
CharStyle *current_style_; // current font, attrib, color..
OutFlags oflags_; // output translation flags (CR_TO_LF, LF_TO_CR, LF_TO_CRLF)

RingBuffer ring_; // terminal history/display ring buffer
Cursor cursor_; // terminal cursor (position, color, etc)
Margin margin_; // terminal margins (top, left, bottom, right)
EscapeSeq escseq; // Escape sequence parsing (ESC [ xterm/vt100)
bool show_unknown_; // if true, show unknown chars as '¿' (default off)
int tabstops_size_; // size of tabstops[] array / be a class *TabStops*.
int tabstops[]; // array of tab stops (0][1) _\-- TODO: This should probably
int autoscroll_dir_; // 0=autoscroll timer off, 3=scrolling up, 4=scrolling down
int autoscroll_amt_; // #pixels above or below edge, used for autoscroll speed
RedrawStyle redraw_style_; // NO_REDRAW, RATE_LIMITED, PER_WRITE
float redraw_rate_; // maximum redraw rate in seconds, default=0.10
bool redraw_modified_; // display modified; used by update_cb() to rate limit redraws
bool redraw_timer_; // if true, redraw timer is running

protected:

const Utf8Char * u8c_ring_row(int grow) const;
const Utf8Char * u8c_hist_row(int hrow) const;
const Utf8Char * u8c_hist_use_row(int hrow) const;
const Utf8Char * u8c_disp_row(int grow) const;

const Utf8Char * u8c_cursor(void);

private:

void create_ring(int drows, int dcols, int hrows);
void init_(int X, int Y, int W, int H, const char *L, int rows, int cols, int hist,bool fontsize_defer);

protected:

int scrollbar_width(void) const;

private:

void create_ring(int drows, int dcols, int hrows);

void init_(int X, int Y, int W, int H, const char *L, int rows, int cols, int hist,bool fontsize_defer);

protected:

int scrollbar_width(void) const;

private:

void create_ring(int drows, int dcols, int hrows);

void init_(int X, int Y, int W, int H, const char *L, int rows, int cols, int hist,bool fontsize_defer);

protected:

int scrollbar_width(void) const;

private:

void create_ring(int drows, int dcols, int hrows);

void init_(int X, int Y, int W, int H, const char *L, int rows, int cols, int hist,bool fontsize_defer);

protected:

int scrollbar_width(void) const;

private:

void create_ring(int drows, int dcols, int hrows);

void init_(int X, int Y, int W, int H, const char *L, int rows, int cols, int hist,bool fontsize_defer);

protected:

int scrollbar_width(void) const;

private:

void create_ring(int drows, int dcols, int hrows);

void init_(int X, int Y, int W, int H, const char *L, int rows, int cols, int hist,bool fontsize_defer);
static void scrollbar_cb(Fl_Widget*, void*); // scrollbar manipulation
static void autoscroll_timer_cb(void*); // mouse drag autoscroll
static void scrollbar_cb2(void);
static void redraw_timer_cb(void*); // redraw rate limiting timer
static void redraw_timer_cb2(void);

protected:
const CharStyle& current_style(void) const;
void current_style(const CharStyle& sty);

private:
int x_to_glob_col(int X, int grow, int &gcol) const;
int xy_to_glob_rowcol(int X, int Y, int &grow, int &gcol) const;

protected:
int w_to_col(int W) const;
int h_to_row(int H) const;

private:
bool is_hist_ring_row(int grow) const;
bool is_disp_ring_row(int grow) const;

protected:
int selection_text_len(void) const;

private:
bool is_printable(char c);
bool is_ctrl(char c);
void handle_SGR(void);

public:
void output_translate(Fl_Terminal::OutFlags val);
Fl_Terminal::OutFlags output_translate(void) const;

protected:
void cursorfgcolor(Fl_Color val);
void cursorbgcolor(Fl_Color val);
Fl_Color cursorfgcolor(void) const;
Fl_Color cursorbgcolor(void) const;

protected:
void cursor_up(int count=1, bool do_scroll=false);
void cursor_down(int count=1, bool do_scroll=false);
void cursor_left(int count=1);
void cursor_right(int count=1, bool do_scroll=false);
void cursor_sol(void);
void cursor_eol(void);
void scroll(int rows);
void insert_rows(int count);
void delete_rows(int count);
void insert_char_eol(char c, int drow, int dcol, int rep);
void insert_char(char c, int rep);
void delete_chars(int drow, int dcol, int rep);
void delete_chars(int rep);

public:
void clear();
void clear(Fl_Color val);
void clear(Fl_Color val);

private:
void save_cursor(void);
void restore_cursor(void);

// API: Terminal operations
void cursor_row(int row);
int cursor_row(void) const;
int cursor_col(void) const;

protected:
void handle_lf(void);
void handle_cr(void);
void handle_ctrl(char c);
void handle_SGR(void);

// Output translation
void output_translate(Fl_Terminal::OutFlags val);

private:
void handle_LF(void);
void handle_CR(void);

// Printing
void handle_crlf(char c);
bool is_printable(char c);
bool is_ctrl(char c);
bool is_ctrl(char c);
void handle_SGR(void);
void handle_DECLARA(void);
void handle_escseq(char c);
void display_modified(void);
void clear_char_at_disp(int drow, int dcol);
const Utf8Char* utf8_char_at_disp(int drow, int dcol) const;
const Utf8Char* utf8_char_at_glob(int grow, int gcol) const;
void repeat_char(char c, int rep);
void utf8_cache_clear(void);
void utf8_cache_flush(void);

// API: Character display output
public:

void putchar(const char* text, int len, int drow, int dcol);
void putchar(char c, int drow, int dcol);
void print_char(const char* text, int len=-1);
void print_char(char c);

// API: String display output
public:

void append_utf8(const char* buf, int len=-1);
void append_ascii(const char* s);
void append(const char* s, int len=-1);
void append(const char* s, int len=-1);

protected:

int handle_unknown_char(void);

// Drawing
void draw_row_bg(int grow, int X, int Y) const;
void draw_row(int grow, int Y) const;
void draw_buff(int Y) const;

private:

void handle_selection_autoscroll(void);

// FLTK: draw(), resize(), handle()
void draw(void) FL_OVERRIDE;
void resize(int X,int Y,int W,int H) FL_OVERRIDE;
int handle(int e) FL_OVERRIDE;

protected:

// Don't make these public, but allow internals and derived classes to maintain brevity.

inline int ring_rows(void) const { return ring_.ring_rows(); } |
inline int ring_cols(void) const { return ring_.ring_cols(); } |
inline int ring_srow(void) const { return ring_.ring_srow(); } |
inline int ring_erow(void) const { return ring_.ring_erow(); } |
inline int hist_rows(void) const { return ring_.hist_rows(); } |
inline int hist_cols(void) const { return ring_.hist_cols(); } |
inline int hist_srow(void) const { return ring_.hist_srow(); } |
inline int hist_erow(void) const { return ring_.hist_erow(); } |
inline int hist_use(void) const { return ring_.hist_use(); } |
inline int hist_use_srow(void) const { return ring_.hist_use_srow(); } |
inline int disp_rows(void) const { return ring_.disp_rows(); } |
inline int disp_cols(void) const { return ring_.disp_cols(); } |
inline int disp_srow(void) const { return ring_.disp_srow(); } |
inline int disp_erow(void) const { return ring_.disp_erow(); } |
inline int offset(void) const { return ring_.offset(); } |
// TODO: CLEAN UP WHAT'S PUBLIC, AND WHAT SHOULD BE 'PROTECTED' AND 'PRIVATE'
// Some of the public stuff should, quite simply, "not be".

// API: Terminal features
public:

int scrollbar_size(void) const;
void scrollbar_size(int val);
int scrollbar_actual_size(void) const;

// API: History
int history_rows(void) const;
void history_rows(int val);
int history_use(void) const;

// API: Display
int display_rows(void) const;
void display_rows(int val);
int display_columns(void) const;
void display_columns(int val);

// API: Box
void box(Fl_Boxtype val) { Fl_Group::box(val); update_screen(false); }

// API: Margins
int margin_left(void) const { return margin_.left(); } |
int margin_right(void) const { return margin_.right(); } |
int margin_top(void) const { return margin_.top(); } |
int margin_bottom(void) const { return margin_.bottom(); } |
void margin_left(int val);
void margin_right(int val);
void margin_top(int val);
void margin_bottom(int val);
# API: Text font/size/color

- void textfont(Fl_Font val);
- void textsize(Fl_Fontsize val);
- void textcolor(Fl_Color val);
- void color(Fl_Color val);
- void textfgcolor(Fl_Color val);
- void textbgcolor(Fl_Color val);
- void textfgcolor_default(Fl_Color val);
- void textbgcolor_default(Fl_Color val);

```cpp
Fl_Font textfont(void) const { return current_style_->fontface(); }
Fl_Fontsize textsize(void) const { return current_style_->fontsize(); }
Fl_Color color(void) const { return Fl_Group::color(); }
Fl_Color textcolor(void) const { return textfgcolor_default(); }
Fl_Color textfgcolor(void) const { return current_style_->fgcolor(); }
Fl_Color textbgcolor(void) const { return current_style_->bgcolor(); }
Fl_Color textfgcolor_default(void) const { return current_style_->defaultfgcolor(); }
Fl_Color textbgcolor_default(void) const { return current_style_->defaultbgcolor(); }
```

- void textfgcolor_xterm(uchar val);
- void textbgcolor_xterm(uchar val);
- void selectionfgcolor(Fl_Color val) { select_.selectionfgcolor(val); }
- void selectionbgcolor(Fl_Color val) { select_.selectionbgcolor(val); }
- Fl_Color selectionfgcolor(void) const { return select_.selectionfgcolor(); }
- Fl_Color selectionbgcolor(void) const { return select_.selectionbgcolor(); }

## API: Text attrib

- void textattrib(uchar val);
- uchar textattrib() const;

## API: Redraw style/rate

- RedrawStyle redraw_style(void) const;
- void redraw_style(RedrawStyle val);
- private:
- bool is_redraw_style(RedrawStyle val) { return redraw_style_ == val; }

## API: Show unknown/unprintable chars

- bool show_unknown(void) const;
- void show_unknown(bool val);

## API: ANSI sequences

- bool ansil(void) const;
- void ansil(bool val);

## API: Simple Terminal API compatibility

- int history_lines(void) const;
- int history_lines(int val);

## API: printf()

- void printf(const char *fmt, ...);
- void vprintf(const char *fmt, va_list ap);

## Ctor

- Fl_Terminal(int X, int Y, int W, int H, const char *L=0);
- Fl_Terminal(int X, int Y, int W, int H, const char*L=0);
- Fl_TERMINAL(int X, int Y, int W, int H, const char*L=0, int rows, int cols, int hist);

## Dtor

- ~Fl_Terminal(void);

## Debugging features

- //DEBUG void show_ring_info() const { ring_.show_ring_info(); };
- //DEBUG void write_row(FILE *fp, Utf8Char *u8c, int cols) const;
- //DEBUG void show_buffers(RingBuffer *a, RingBuffer *b=0) const;
- #endif
/* Doxygen can’t find <FL/fl_attr.h> */

#include "fl_attr.h"

#ifndef ASSERT_UTF8
#define ASSERT_UTF8
#endif

/*
character size" is the size of a UTF-8 character in bytes
"character width" is the width of a Unicode character in pixels
"column" was originally defined as a character offset from the left margin.
It was identical to the byte offset. In UTF-8, we have neither a byte offset
nor truly fixed width fonts [*]. Column could be a pixel value multiplied with
an average character width (which is a bearable approximation).

* in Unicode, there are no fixed width fonts! Even if the ASCII characters may
happen to be all the same width in pixels, Chinese characters surely are not.
There are plenty of exceptions, like ligatures, that make special handling of
"fixed" character widths a nightmare. I decided to remove all references to
fixed fonts and see "columns" as a multiple of the average width of a
character in the main font.
- Matthias
*/

/* Maximum length in characters of a tab or control character expansion
of a single buffer character */

#define FL_TEXT_MAX_EXP_CHAR_LEN 20

class Fl_Text_Undo_Action_List;
class Fl_Text_Undo_Action;

class FL_EXPORT Fl_Text_Selection {
friend class Fl_Text_Buffer;
public:
void set(int startpos, int endpos);

void update(int pos, int nDeleted, int nInserted);

int start() const { return mSelected ? mStart : 0; }

int end() const { return mSelected ? mEnd : 0; }

bool selected() const { return mSelected; }

void selected(bool b) { mSelected = b; }

int length() const { return mSelected ? mEnd - mStart : 0; }

int includes(int pos) const;

int selected(int *startpos, int *endpos) const;
position(int *startpos, int *endpos) const {
  return selected(startpos, endpos); }
protected:

int mStart;
int mEnd;
bool mSelected;
};
class FL_EXPORT Fl_Text_Buffer {
public:

// Sets the selection range and selected().
void set(int startpos, int endpos);

// Updates a selection after text was modified.
void update(int pos, int nDeleted, int nInserted);

int start() const { return mSelected ? mStart : 0; }

int end() const { return mSelected ? mEnd : 0; }

bool selected() const { return mSelected; }

void selected(bool b) { mSelected = b; }

int length() const { return mSelected ? mEnd - mStart : 0; }

// Returns true if position \p pos is in this Fl_Text_Selection.
int includes(int pos) const;

// Returns true if selected() and the positions of this selection.
int selected(int *startpos, int *endpos) const;

int position(int *startpos, int *endpos) const {
  return selected(startpos, endpos); }
protected:

int mStart;
int mEnd;
bool mSelected;
};
typedef void (*FL_Text_Modify_Callback)(int pos, int nInserted, int nDeleted,
  int nRestyled, const char * deletedText,
  void * cbArg);
typedef void (*FL_Text_Predelete_Callback)(int pos, int nDeleted, void * cbArg);
class FL_EXPORT Fl_Text_Buffer {
public:

 Fl_Text_Buffer(int requestedSize = 0, int preferredGapSize = 1024);
 ~Fl_Text_Buffer();

 int length() const { return mLength; }

 char* text() const;

 void text(const char* text);

 char* text_range(int start, int end) const;

 unsigned int char_at(int pos) const;
 char byte_at(int pos) const;

 char* address(int pos) { return (pos < mGapStart) ? mBuf+pos : mBuf+pos+mGapEnd-mGapStart; }

 void insert(int pos, const char* text, int insertedLength = -1);
 void append(const char* t, int addedLength = -1) { insert(length(), t, addedLength); }
 void vprintf(const char* fmt, va_list ap);
 void printf(const char* fmt, ...);

 void remove(int start, int end);

 void replace(int start, int end, const char* text, int insertedLength = -1);

 void copy(Fl_Text_Buffer* fromBuf, int fromStart, int fromEnd, int toPos);

 int undo(int* cp=0);

 bool can_undo() const;

 int redo(int* cp=0);

 bool can_redo() const;

 void canUndo(char flag=1);

 int insertfile(const char* file, int pos, int buflen = 128*1024);

 int appendfile(const char* file, int buflen = 128*1024)
 { return insertfile(file, length(), buflen); }

 int loadfile(const char* file, int buflen = 128*1024)
 { select(0, length()); remove_selection(); return appendfile(file, buflen); }

 int outputfile(const char* file, int start, int end, int buflen = 128*1024);

 int savefile(const char* file, int buflen = 128*1024)
 { return outputfile(file, 0, length(), buflen); }

 int tab_distance() const { return mTabDist; }

 void tab_distance(int tabDist);

 void select(int start, int end);

 int selected() const { return mPrimary.selected(); }

 void unselect();

 int selection_position(int* start, int* end);

 char* selection_text();

 void remove_selection();

 void replace_selection(const char* text);

 void secondary_select(int start, int end);

 int secondary_selected() { return mSecondary.selected(); }

 void secondary_unselect();

 int secondary_selection_position(int* start, int* end);

 char* secondary_selection_text();
void remove_secondary_selection();

void replace_secondary_selection(const char* text);

void highlight(int start, int end);

int highlight() { return mHighlight.selected(); }

void unhighlight();

int highlight_position(int start, int end);

int highlight_text();

void add_modify_callback(Fl_Text_Modify_Cb bufModifiedCB, void* cbArg);

void remove_modify_callback(Fl_Text_Modify_Cb bufModifiedCB, void* cbArg);

void call_modify_callbacks() { call_modify_callbacks(0, 0, 0, 0, 0); }

void add_predelete_callback(Fl_Text_Predelete_Cb bufPredelCB, void* cbArg);

void remove_predelete_callback(Fl_Text_Predelete_Cb predelCB, void* cbArg);

void call_predelete_callbacks() { call_predelete_callbacks(0, 0); }

char* line_text(int pos) const;

int line_start(int pos) const;

int line_end(int pos) const;

int word_start(int pos) const;

int word_end(int pos) const;

int count_displayed_characters(int lineStartPos, int targetPos) const;

int skip_displayed_characters(int lineStartPos, int nChars);

int count_lines(int startPos, int endPos) const;

int skip_lines(int startPos, int nLines);

int rewind_lines(int startPos, int nLines);

int findchar_forward(int startPos, unsigned searchChar, int* foundPos) const;

int findchar_backward(int startPos, unsigned int searchChar, int* foundPos) const;

int search_forward(int startPos, const char* searchString, int* foundPos,
                   int matchCase = 0) const;

int search_backward(int startPos, const char* searchString, int* foundPos,
                     int matchCase = 0) const;

const Fl_Text_Selection* primary_selection() const { return &mPrimary; }

Fl_Text_Selection* primary_selection() { return &mPrimary; }

const Fl_Text_Selection* secondary_selection() const { return &mSecondary; }

const Fl_Text_Selection* highlight_selection() const { return &mHighlight; }

int prev_char(int ix) const;

int prev_char_clipped(int ix) const;

int next_char(int ix) const;

int next_char_clipped(int ix) const;

int utf8_align(int) const;

int input_file_was_transcoded;

static const char* file_encoding_warning_message;

void (*transcoding_warning_action)(Fl_Text_Buffer*);

bool is_word_separator(int pos) const;

protected:

void call_modify_callbacks(int pos, int nDeleted, int nInserted,
                           int nRestyled, const char* deletedText) const;

void call_predelete_callbacks(int pos, int nDeleted) const;
int insert_(int pos, const char* text, int insertedLength = -1);

void remove_(int start, int end);

void redisplay_selection(Fl_Text_Selection* oldSelection, 
                        Fl_Text_Selection* newSelection) const;

void move_gap(int pos);

void reallocate_with_gap(int newGapStart, int newGapLen);

char* selection_text_(Fl_Text_Selection* sel) const;

void remove_selection_(Fl_Text_Selection* sel);

void replace_selection_(Fl_Text_Selection* sel, const char* text);

void update_selections(int pos, int nDeleted, int nInserted);

int apply_undo(Fl_Text_Undo_Action* action, int* cursorPos);

Fl_Text_Selection mPrimary;
Fl_Text_Selection mSecondary;
Fl_Text_Selection mHighlight;
int mLength;
char* mBuf;
int mGapStart;
int mGapEnd;

// The hardware tab distance used by all displays for this buffer,
// and used in computing offsets for rectangular selection operations.
int mTabDist;

// mModifyProcs;
void* mCBArg;
int mPredeleteProcs;
int mPredeleteCBArg;
int mCursorPosHint;
char mCanUndo;
int mPreferredGapSize;

Fl_Text_Undo_Action* mUndo;
Fl_Text_Undo_Action_List* mUndoList;
Fl_Text_Undo_Action_List* mRedoList;
};
#endif

---

13.151 Fl_Text_Display.H

/*
file Fl_Text_Displ
enum { NORMAL_CURSOR, CARET_CURSOR, DIM_CURSOR, BLOCK_CURSOR, HEAVY_CURSOR, SIMPLE_CURSOR };

enum { CURSOR_POS, CHARACTER_POS }; 

enum { DRAG_NONE = -2, DRAG_START_DND = -1, DRAG_CHAR = 0, DRAG_WORD = 1, DRAG_LINE = 2 }; 

enum { WRAP_NONE, WRAP_AT_COLUMN, WRAP_AT_PIXEL, WRAP_AT_BOUNDS }; 

friend int fl_text_drag_prepare(int pos, int key, Fl_Text_Display* d); 
friend void fl_text_drag_me(int pos, Fl_Text_Display* d); 

typedef void (*Unfinished_Style_Cb)(int, void *); 

struct Style_Table_Entry { Fl_Color color; Fl_Font font; Fl_Fontsize size; unsigned attr; Fl_Color bgcolor; }; 

enum { ATTR_BGCOLOR = 0x0001, ATTR_BGCOLOR_EXT_ = 0x0002, ATTR_BGCOLOR_EXT = 0x0003, ATTR_UNDERLINE = 0x0004, ATTR_GRAMMAR = 0x0008, ATTR_SPELLING = 0x000C, ATTR_STRIKE_THROUGH = 0x0010, ATTR_LINES_MASK = 0x001C }; 

Fl_Text_Display(int X, int Y, int W, int H, const char *l = 0); ~Fl_Text_Display(); 
int handle(int e) FL_OVERRIDE; 
void buffer(Fl_Text_Buffer* buf); 
void buffer(Fl_Text_Buffer& buf) { buffer(&buf); } 
Fl_Text_Buffer* buffer() const { return mBuffer; } 
Fl_Text_Buffer* style_buffer() const { return mStyleBuffer; } 
void redisplay_range(int start, int end); 
void scroll(int topLineNum, int horizOffset); 
void insert(const char* text); 
void overstrike(const char* text); 
void insert_position(int newPos); 
int insert_position() const { return mCursorPos; } 
int position_to_xy(int pos, int* x, int* y) const; 
int in_selection(int x, int y) const; 
void show_insert_position(); 
int move_right(); 
int move_left(); 
int move_up(); 
int move_down(); 
int count_lines(int start, int end, bool start_pos_is_line_start) const; 
int line_start(int pos) const; 
int line_end(int startPos, bool startPosIsLineStart) const; 
int skip_lines(int startPos, int nLines, bool startPosIsLineStart); 
int rewind_lines(int startPos, int nLines); 
void next_word(void); 
void previous_word(void); 

Generated by Doxygen
void show_cursor(int b = 1);
void hide_cursor() { show_cursor(0); }
void cursor_style(int style);
int cursor_style() const { return mCursorStyle; }
Fl_Color cursor_color() const {return mCursor_color;}
int scrollbar_width() const {
    return scrollbar_width_ ? scrollbar_width_ : Fl::scrollbar_size();
}
void scrollbar_width(int width) {
    Fl::scrollbar_size(width);
    scrollbar_width_ = 0;
}
int scrollbar_size() const {
    return scrollbar_width;;
}
void scrollbar_size(int newSize) {
    scrollbar_width_ = newSize;
}
Fl.Align scrollbar_align() const { return scrollbar_align_; }
void scrollbar_align(Fl.Align a) { scrollbar_align_ = a; }
int word_start(int pos) const { return buffer()->word_start(pos); }
int word_end(int pos) const { return buffer()->word_end(pos); }
int shortcut() const {return shortcut_;
void shortcut(int s) {shortcut_ = s;
Fl_Font textfont() const {return textfont_;
void textfont(Fl_Font s) {textfont_ = s; mColumnScale = 0;
Fl_Fontsize textsize() const {return textsize_;
void textsize(Fl_Fontsize s) {textsize_ = s; mColumnScale = 0;
Fl_Color textcolor() const {return textcolor_;
void textcolor(Fl_Color n) {textcolor_ = n;
Fl_Color grammar_underline_color(Fl_Color color) { grammar_underline_color_ = color; }
Fl_Color spelling_underline_color(Fl_Color color) { spelling_underline_color_ = color; }
Fl_Color secondary_selection_color(Fl_Color color) { secondary_selection_color_ = color; }
int wrapped_column(int row, int column) const;
int wrapped_row(int row) const;
void wrap_mode(int wrap, int wrap_margin);
virtul void recalc_display();
void resize(int X, int Y, int W, int H) Fl_OVERRIDE;
double x_to_col(double x) const;
double col_to_x(double col) const;
void linenumber_width(int width);
int linenumber_width() const;
void linenumber_font(Fl_Font val);
void linenumber_font() const;
void linenumber_size(Fl_Fontsize val);
Fl_Fontsize linenumber_size() const;
void linenumber_fgcolor(Fl_Color val);
Fl_Color linenumber_fgcolor() const;
void linenumber_bgcolor(Fl_Color val);
Fl_Color linenumber_bgcolor() const;
void linenumber_align(Fl_Align val);
Fl_Align linenumber_align() const;
void linenumber_format(const char * val);
const char * linenumber_format() const;

protected:
// Most (all?) of this stuff should only be called from resize() or
// draw().
// Anything with "vline" indicates that it deals with currently
// visible lines.
void draw() FL_OVERRIDE;
vvoid draw_text(int X, int Y, int W, int H);
vvoid draw_range(int start, int end);
vvoid draw_cursor(int, int);
vvoid draw_string(int style, int x, int y, int toX, const char *string,
int nChars) const;
vvoid draw_vline(int style, int x, int y, int toX, const char *string,
int nChars) const;
vvoid draw_vline(int visLineNum, int leftClip, int rightClip,
int leftCharIndex, int rightCharIndex);
int find_y(const char *s, int len, int style, int x) const;
enum {
DRAW_LINE,
FIND_INDEX,
FIND_INDEX_FROM_ZERO,
GET_WIDTH,
FIND_CURSOR_INDEX
};
int handle_vline(int mode,
int lineStart, int lineLen, int leftChar, int rightChar,
int topClip, int bottomClip,
int leftClip, int rightClip) const;
int handle_rmb(int readonly);
void draw_line_numbers(bool clearAll);
void clear_rect(int style, int x, int y, int width, int height) const;
void display_insert();
void offset_line_starts(int newTopLineNum);
void calc_line_starts(int startLine, int endLine);
void update_line_starts(int pos, int charsInserted, int charsDeleted,
int linesInserted, int linesDeleted, int *scrolled);
void calc_last_char();
int position_to_line(int pos, int * lineNum) const;
double string_width(const char * string, int length, int style) const;
static void scroll_timer_cb(void *);
static void buffer_predelete_cb(int pos, int nDeleted, void * cbArg);
static void buffer_modified_cb(int pos, int nInserted, int nDeleted,
int nRestyled, const char * deletedText,
void * cbArg);
static void h_scrollbar_cb(Fl_Scrollbar * w, Fl_Text_Display* d);
static void v_scrollbar_cb(Fl_Scrollbar * w, Fl_Text_Display* d);
void update_v_scrollbar();
int measure_vline(int visLineNum) const;
int longest_vline() const;
int empty_vlines() const;
int vline_length(int visLineNum) const;
int xy_to_position(int x, int y, int PosType = CHARACTER_POS) const;
void xy_to_rowcol(int x, int y, int row, int column,
int PosType = CHARACTER_POS) const;
void maintain_absolute_top_line_number(int state);
int get_absolute_top_line_number() const;
void absolute_top_line_number(int oldFirstChar);
int maintaining_absolute_top_line_number() const;

Generated by Doxygen
void reset_absolute_top_line_number();

int position_to_linecol(int pos, int* lineNum, int* column) const;

int scroll_(int topLineNum, int horizOffset);

void extend_range_for_styles(int* start, int* end);

void find_wrap_range(const char *deletedText, int pos, int nInserted,
int nDeleted, int *modRangeStart, int *modRangeEnd,
int *linesInserted, int *linesDeleted);

int wrap_(int topLineNum, int horizOffset);

int measure_deleted_lines(int pos, int nDeleted);

void wrapped_line_counter(Fl_Text_Buffer* buf, int startPos, int maxPos,
maxLines, bool startFosIsLineStart,
int styleBufOffset, int *retPos, int *retLines,
int *retLineStart, int *retLineEnd,
bool countLastLineMissingNewline = true) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char *s, int colNum, int pos) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char* deletedText, int pos, int nInserted,
int nDeleted, int *modRangeStart, int *modRangeEnd,
int *linesInserted, int *linesDeleted);

void measure_deleted_lines(int pos, int nDeleted);

void find_wrap_range(const char *deletedText, int pos, int nInserted,
int nDeleted, int *modRangeStart, int *modRangeEnd,
int *linesInserted, int *linesDeleted);

void find_wrap_range(const char* deletedText, int pos, int nInserted,
int nDeleted, int* modRangeStart, int* modRangeEnd,
int* linesInserted, int* linesDeleted);

int wrap_(int topLineNum, int horizOffset);

int measure_deleted_lines(int pos, int nDeleted);

void wrapped_line_counter(Fl_Text_Buffer* buf, int startPos, int maxPos,
maxLines, bool startFosIsLineStart,
int styleBufOffset, int* retPos, int* retLines,
int* retLineStart, int* retLineEnd,
bool countLastLineMissingNewline = true) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char* s, int colNum, int pos) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char* deletedText, int pos, int nInserted,
int nDeleted, int* modRangeStart, int* modRangeEnd,
int* linesInserted, int* linesDeleted);

void measure_deleted_lines(int pos, int nDeleted);

void find_wrap_range(const char* deletedText, int pos, int nInserted,
int nDeleted, int* modRangeStart, int* modRangeEnd,
int* linesInserted, int* linesDeleted);

int wrap_(int topLineNum, int horizOffset);

int measure_deleted_lines(int pos, int nDeleted);

void wrapped_line_counter(Fl_Text_Buffer* buf, int startPos, int maxPos,
maxLines, bool startFosIsLineStart,
int styleBufOffset, int* retPos, int* retLines,
int* retLineStart, int* retLineEnd,
bool countLastLineMissingNewline = true) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char* s, int colNum, int pos) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char* deletedText, int pos, int nInserted,
int nDeleted, int* modRangeStart, int* modRangeEnd,
int* linesInserted, int* linesDeleted);

void measure_deleted_lines(int pos, int nDeleted);

void find_wrap_range(const char* deletedText, int pos, int nInserted,
int nDeleted, int* modRangeStart, int* modRangeEnd,
int* linesInserted, int* linesDeleted);

int wrap_(int topLineNum, int horizOffset);

int measure_deleted_lines(int pos, int nDeleted);

void wrapped_line_counter(Fl_Text_Buffer* buf, int startPos, int maxPos,
maxLines, bool startFosIsLineStart,
int styleBufOffset, int* retPos, int* retLines,
int* retLineStart, int* retLineEnd,
bool countLastLineMissingNewline = true) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char* s, int colNum, int pos) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

void find_wrap_range(const char* deletedText, int pos, int nInserted,
int nDeleted, int* modRangeStart, int* modRangeEnd,
int* linesInserted, int* linesDeleted);

int wrap_(int topLineNum, int horizOffset);

int measure_deleted_lines(int pos, int nDeleted);

void wrapped_line_counter(Fl_Text_Buffer* buf, int startPos, int maxPos,
maxLines, bool startFosIsLineStart,
int styleBufOffset, int* retPos, int* retLines,
int* retLineStart, int* retLineEnd,
bool countLastLineMissingNewline = true) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char* s, int colNum, int pos) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

void find_wrap_range(const char* deletedText, int pos, int nInserted,
int nDeleted, int* modRangeStart, int* modRangeEnd,
int* linesInserted, int* linesDeleted);

int wrap_(int topLineNum, int horizOffset);

int measure_deleted_lines(int pos, int nDeleted);

void wrapped_line_counter(Fl_Text_Buffer* buf, int startPos, int maxPos,
maxLines, bool startFosIsLineStart,
int styleBufOffset, int* retPos, int* retLines,
int* retLineStart, int* retLineEnd,
bool countLastLineMissingNewline = true) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char* s, int colNum, int pos) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

void find_wrap_range(const char* deletedText, int pos, int nInserted,
int nDeleted, int* modRangeStart, int* modRangeEnd,
int* linesInserted, int* linesDeleted);

int wrap_(int topLineNum, int horizOffset);

int measure_deleted_lines(int pos, int nDeleted);

void wrapped_line_counter(Fl_Text_Buffer* buf, int startPos, int maxPos,
maxLines, bool startFosIsLineStart,
int styleBufOffset, int* retPos, int* retLines,
int* retLineStart, int* retLineEnd,
bool countLastLineMissingNewline = true) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char* s, int colNum, int pos) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

void find_wrap_range(const char* deletedText, int pos, int nInserted,
int nDeleted, int* modRangeStart, int* modRangeEnd,
int* linesInserted, int* linesDeleted);

int wrap_(int topLineNum, int horizOffset);

int measure_deleted_lines(int pos, int nDeleted);

void wrapped_line_counter(Fl_Text_Buffer* buf, int startPos, int maxPos,
maxLines, bool startFosIsLineStart,
int styleBufOffset, int* retPos, int* retLines,
int* retLineStart, int* retLineEnd,
bool countLastLineMissingNewline = true) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char* s, int colNum, int pos) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

void find_wrap_range(const char* deletedText, int pos, int nInserted,
int nDeleted, int* modRangeStart, int* modRangeEnd,
int* linesInserted, int* linesDeleted);

int wrap_(int topLineNum, int horizOffset);

int measure_deleted_lines(int pos, int nDeleted);

void wrapped_line_counter(Fl_Text_Buffer* buf, int startPos, int maxPos,
maxLines, bool startFosIsLineStart,
int styleBufOffset, int* retPos, int* retLines,
int* retLineStart, int* retLineEnd,
bool countLastLineMissingNewline = true) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char* s, int colNum, int pos) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

void find_wrap_range(const char* deletedText, int pos, int nInserted,
int nDeleted, int* modRangeStart, int* modRangeEnd,
int* linesInserted, int* linesDeleted);

int wrap_(int topLineNum, int horizOffset);

int measure_deleted_lines(int pos, int nDeleted);

void wrapped_line_counter(Fl_Text_Buffer* buf, int startPos, int maxPos,
maxLines, bool startFosIsLineStart,
int styleBufOffset, int* retPos, int* retLines,
int* retLineStart, int* retLineEnd,
bool countLastLineMissingNewline = true) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char* s, int colNum, int pos) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

void find_wrap_range(const char* deletedText, int pos, int nInserted,
int nDeleted, int* modRangeStart, int* modRangeEnd,
int* linesInserted, int* linesDeleted);

int wrap_(int topLineNum, int horizOffset);

int measure_deleted_lines(int pos, int nDeleted);

void wrapped_line_counter(Fl_Text_Buffer* buf, int startPos, int maxPos,
maxLines, bool startFosIsLineStart,
int styleBufOffset, int* retPos, int* retLines,
int* retLineStart, int* retLineEnd,
bool countLastLineMissingNewline = true) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;

double measure_proportional_character(const char* s, int colNum, int pos) const;

void find_line_end(int pos, bool start_pos_is_line_start, int* lineEnd,
int* nextLineStart) const;
Fl_Color mCursor_color;
Fl_Scrollbar * mHScrollBar;
Fl_Scrollbar * mVScrollBar;
int scrollbar_width_; // size of scrollbar trough (behavior changed in 1.4)
Fl_Align scrollbar_align_;
int dragPos, dragType, dragging;
struct { int x, y, w, h; } text_area;
int shortcut_;
Fl_Font textfont_;
Fl_Fontsize textsize_;
Fl_Color textcolor_;
Fl_Color grammar_underline_color_; 
Fl_Color spelling_underline_color_; 
Fl_Color secondary_selection_color_; 
const char * linenumber_format_; 
}
};
#endif

13.152 Fl_Text_Editor.H

/* File
Fl_Text_Editor widget */

#ifndef FL_TEXT_EDITOR_H
#define FL_TEXT_EDITOR_H

#include "Fl_Text_Display.H"

#define FL_TEXT_EDITOR_ANY_STATE (-1L)

class FL_EXPORT Fl_Text_Editor : public Fl_Text_Display {
public:

typedef int (*Key_Func)(int key, Fl_Text_Editor* editor);

struct Key_Binding {
  int key;
  int state;
  Key_Func function;
  Key_Binding* next;
};

Fl_Text_Editor(int X, int Y, int W, int H, const char * l = 0);
~Fl_Text_Editor() { remove_all_key_bindings(); }
int handle(int e) FL_OVERRIDE;

void insert_mode(int b) { insert_mode_ = b; }
int insert_mode() { return insert_mode_; }
void tab_nav(int val);
int tab_nav() const;

};
#endif
void add_key_binding(int key, int state, Key_Func f, Key_Binding** list);
void add_key_binding(int key, int state, Key_Func f);
void remove_key_binding(int key, int state, Key_Binding** list);
void remove_key_binding(int key, int state);
void remove_all_key_bindings(Key_Binding** list);
void remove_all_key_bindings();
void add_default_key_bindings(Key_Binding** list);
Key_Func bound_key_function(int key, int state, Key_Binding* list) const;
Key_Func bound_key_function(int key, int state) const;
return bound_key_function(key, state, key_bindings);
void default_key_function(Key_Func f) { default_key_function_ = f; }

// functions for the built in default bindings
static int kf_default(int c, Fl_Text_Editor* e);
static int kf_ignore(int c, Fl_Text_Editor* e);
static int kf_backspace(int c, Fl_Text_Editor* e);
static int kf_move(int c, Fl_Text_Editor* e);
static int kf_ctrl_move(int c, Fl_Text_Editor* e);
static int kf_ctrl_a_move(int c, Fl_Text_Editor* e);
static int kf_home(int c, Fl_Text_Editor* e);
static int kf_end(int c, Fl_Text_Editor* e);
static int kf_left(int c, Fl_Text_Editor* e);
static int kf_up(int c, Fl_Text_Editor* e);
static int kf_right(int c, Fl_Text_Editor* e);
static int kf_down(int c, Fl_Text_Editor* e);
static int kf_page_up(int c, Fl_Text_Editor* e);
static int kf_page_down(int c, Fl_Text_Editor* e);
static int kf_delete(int c, Fl_Text_Editor* e);
static int kf_copy(int c, Fl_Text_Editor* e);
static int kf_cut(int c, Fl_Text_Editor* e);
static int kf_paste(int c, Fl_Text_Editor* e);
static int kf_select_all(int c, Fl_Text_Editor* e);
static int kf_undo(int c, Fl_Text_Editor* e);
static int kf_redo(int c, Fl_Text_Editor* e);
protected:
int handle_key();
void maybe_do_callback(Fl_Callback_Reason reason = FL_REASON_CHANGED);

// for the built in default bindings
static Key_Binding* global_key_bindings;

// for the built in default bindings
static Key_Func default_key_function_;
The Fl_Tile class lets you resize its children by dragging the border between them.

```cpp
class FL_EXPORT Fl_Tile : public Fl_Group {
public:
    int handle(int event) FL_OVERRIDE;
    Fl_Tile(int X, int Y, int W, int H, const char *L=0);
    ~Fl_Tile() FL_OVERRIDE;
    void resize(int X, int Y, int W, int H) FL_OVERRIDE;
    virtual void move_intersection(int oldx, int oldy, int newx, int newy);
    virtual void drag_intersection(int oldx, int oldy, int newx, int newy);
    FL_DEPRECATED("in 1.4.0 - use move_intersection(p) instead",
        void position(int oldx, int oldy, int newx, int newy)) { move_intersection(oldx, oldy, newx, newy); }
    void position(int x, int y) { Fl_Group::position(x, y); }
    void size_range(int index, int minw, int minh, int maxw=0x7FFFFFFF, int maxh=0x7FFFFFFF);
    void size_range(Fl_Widget *w , int minw, int minh, int maxw=0x7FFFFFFF, int maxh=0x7FFFFFFF);
    void init_size_range(int default_min_w = -1, int default_min_h = -1);
protected:
    int cursor_;  // set one of n (0..3) cursors
    Fl_Cursor *cursors_;    // set one of n (0..3) cursors
    typedef struct { int minw, minh, maxw, maxh; } Size_Range;
    Size_Range *size_range_;  // set one of n (0..3) cursors
    int size_range_size_, size_range_capacity_;  // set one of n (0..3) cursors
    int default_min_w_, default_min_h_;  // set one of n (0..3) cursors
    int on_insert(Fl_Widget *, int) FL_OVERRIDE;
    int on_move(int, int) FL_OVERRIDE;
    void on_remove(int) FL_OVERRIDE;
};
#endif
```

13.154 Fl_Tiled_Image.H

```cpp
class FL_EXPORT Fl_Tiled_Image : public Fl_Image {
protected:
    Fl_Cursor *cursors_;    // set one of n (0..3) cursors
    typedef struct { int minw, minh, maxw, maxh; } Size_Range;
    Size_Range *size_range_;  // set one of n (0..3) cursors
    int size_range_size_, size_range_capacity_;  // set one of n (0..3) cursors
    int default_min_w_, default_min_h_;  // set one of n (0..3) cursors
    void on_insert(Fl_Widget *, int) FL_OVERRIDE;
    int on_move(int, int) FL_OVERRIDE;
    void on_remove(int) FL_OVERRIDE;
};
#endif
```
13.155 Fl_Timer.H

```c
Fl_Tiled_Image(Fl_Image *i, int W = 0, int H = 0);
virtual ~Fl_Tiled_Image();
Fl_Image *copy(int W, int H) const FL_OVERRIDE;
Fl_Image *copy() const {
    return Fl_Image::copy();
}
void color_average(Fl_Color c, float i) FL_OVERRIDE;
void desaturate() FL_OVERRIDE;
void draw(int X, int Y, int W, int H, int cx = 0, int cy = 0) FL_OVERRIDE;
void draw(int X, int Y) {draw(X, Y, w(), h(), 0, 0);}
Fl_Image *image() { return image_; }
```

```c
#endif // !Fl_Tiled_Image_H
```

13.155 Fl_Timer.H

```c
// Timer header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2010 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// Fl_Timer widget .
```

```c
#ifndef Fl_Timer_H
#define Fl_Timer_H
```

```c
#ifndef Fl_Widget_H
#include "Fl_Widget.H"
```

```c
// values for type():
#define FL_NORMAL_TIMER 0
#define FL_VALUE_TIMER 1
#define FL_HIDDEN_TIMER 2
```

```c
class FL_EXPORT Fl_Timer : public Fl_Widget {
static void stepcb(void *);
void step();
char on, direction_;
double delay, total;
long lastsec, lastusec;
```

```c
protected:
void draw() FL_OVERRIDE;
```

```c
void handle(int) FL_OVERRIDE;
Fl_Timer(uchar t,int x,int y,int w,int h, const char *l);
~Fl_Timer();
void value(double);
double value() const {return delay>0.0?delay:0.0;}
char direction() const {return direction_;}
void direction(char d) {direction_ = d;}
char suspended() const {return !on;}
void suspended(char d);
```

```c
};
#endif // Fl_Timer_H
```

13.156 Fl_Toggle_Button.H

```c
// Toggle button header file for the Fast Light Tool Kit (FLTK).
```

```c
#ifndef Fl_Toggle_Button_H
#define Fl_Toggle_Button_H
```

```c
#include "Fl_Widget.H"
```

```c
// values for type():
#define FL_TOGGLE 0
#define FL_RADIO 1
```

```c
class FL_EXPORT Fl_Toggle_Button : public Fl_Widget {
static void stepcb(void *);
void step();
```

```c
void handle(int) FL_OVERRIDE;
```

```c
int X, int Y, int W, int H, int cx = 0, int cy = 0)
```

```c
void draw(int X, int Y, int W, int H, int cx = 0, int cy = 0) {
    draw(X, Y, w(), h(), 0, 0);}
```

Generated by Doxygen
/*
    Fl_Toggle_Button widget . */

#ifndef Fl_Toggle_Button_H
#define Fl_Toggle_Button_H

#include "Fl_Button.H"

class FL_EXPORT Fl_Toggle_Button : public Fl_Button {
public:
    Fl_Toggle_Button(int X, int Y, int W, int H, const char *l=0);
};
#endif

13.157  Fl_Toggle_Light_Button.H

#ifndef Fl_Toggle_Light_Button
#include "Fl_Light_Button.H"
#define Fl_Toggle_Light_Button Fl_Light_Button
#endif

13.158  Fl_Toggle_Round_Button.H

#ifndef Fl_Toggle_Round_Button
#include "Fl_Round_Button.H"
#define Fl_Toggle_Round_Button Fl_Round_Button
#endif

13.159  Fl_Tooltip.H

#ifndef Fl_Tooltip
#include "Fl_Tooltip.H"
#define Fl_Tooltip Fl_Tooltip
#endif

Generated by Doxygen
13.160 Fl_Tree.H File Reference

This file contains the definitions of the Fl_Tree class.
#include <FL/Fl.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Scrollbar.H>
#include <FL/fl_draw.H>
#include <FL/Fl_Tree_Item.H>
#include <FL/Fl_Tree_Prefs.H>

## Classes
- **class Fl_Tree**
  
  Tree widget.

## Enumerations
- **enum Fl_Tree_Reason**
  
  The reason the callback was invoked.

  - **FL_TREE_REASON_NONE** = FL_REASON_UNKNOWN
  - **FL_TREE_REASON_SELECTED**
  - **FL_TREE_REASON_DESELECTED** = FL_REASON_DESELECTED
  - **FL_TREE_REASON_RESELECTED** = FL_REASON_RESELECTED
  - **FL_TREE_REASON_OPENED** = FL_REASON_OPENED
  - **FL_TREE_REASON_CLOSED** = FL_REASON_CLOSED
  - **FL_TREE_REASON_DRAGGED** = FL_REASON_DRAGGED

  The reason the callback was invoked.

### 13.160.1 Detailed Description
This file contains the definitions of the Fl_Tree class.

### 13.160.2 Enumeration Type Documentation
#### 13.160.2.1 Fl_Tree_Reason

**enum Fl_Tree_Reason**
The reason the callback was invoked.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_REASON_NONE</td>
<td>unknown reason</td>
</tr>
<tr>
<td>FL_TREE_REASON_SELECTED</td>
<td>an item was selected</td>
</tr>
<tr>
<td>FL_TREE_REASON_DESELECTED</td>
<td>an item was de-selected</td>
</tr>
<tr>
<td>FL_TREE_REASON_RESELECTED</td>
<td>an item was re-selected (double-clicked). See Fl_Tree_Item_Reselect_Mode to</td>
</tr>
<tr>
<td></td>
<td>enable this.</td>
</tr>
<tr>
<td>FL_TREE_REASON_OPENED</td>
<td>an item was opened</td>
</tr>
<tr>
<td>FL_TREE_REASON_CLOSED</td>
<td>an item was closed</td>
</tr>
<tr>
<td>FL_TREE_REASON_DRAGGED</td>
<td>an item was dragged into a new place</td>
</tr>
</tbody>
</table>

### 13.161 Fl_Tree.H

Go to the documentation of this file.
enum Fl_Tree_Reason {
    FL_TREE_REASON_NONE = FL_REASON_UNKNOWN,
    FL_TREE_REASON_SELECTED = FL_REASON_SELECTED,
    FL_TREE_REASON_DESELECTED = FL_REASON_DESELECTED,
    FL_TREE_REASON_RESELECTED = FL_REASON_RESELECTED,
    FL_TREE_REASON_OPENED = FL_REASON_OPENED,
    FL_TREE_REASON_CLOSED = FL_REASON_CLOSED,
    FL_TREE_REASON_DRAGGED = FL_REASON_DRAGGED
};

class FL_EXPORT Fl_Tree : public Fl_Group {
friend class Fl_Tree_Item;

Fl_Tree_Item _root; // can be null!
Fl_Tree_Item _item_focus; // item that has focus box
Fl_Tree_Item _callback_item; // item invoked during callback (can be NULL)
Fl_Tree_Prefs _prefs; // all the tree's settings
int _scrollbar_size; // size of scrollbar trough
int _tree_w;
int _tree_h;
void fix_scrollbar_order();

protected:
int _vscroll;
int _hscroll;
int _tox,_toy,_tow,_toh;
int _tix,_tiy,_tiw,_tih;

public:
Fl_Tree(int X, int Y, int W, int H, const char *L=0);
~Fl_Tree();
int handle(int e) FL_OVERRIDE;
void show_self();
void resize(int,int,int,int) FL_OVERRIDE;

root methods
void root_label(const char *new_label);
Fl_Tree_Item * root();

Item creation/removal methods
Fl_Tree_Item *add(const char *path, Fl_Tree_Item *newitem=0);
Fl_Tree_Item * add(Fl_Tree_Item *parent_item, const char *name);
Fl_Tree_Item *insert_above(Fl_Tree_Item *above, const char *name);
Fl_Tree_Item * insert(Fl_Tree_Item *item, const char *name, int pos);
int remove(Fl_Tree_Item *item);
void clear();
void clear_children(Fl_Tree_Item *item);

Item lookup methods
Fl_Tree_Item *find_item(const char *path);
const Fl_Tree_Item *find_item(const char *path) const;

Item creation/removal methods
Fl_Tree_Item *add(const char *path, Fl_Tree_Item *newitem=0);
Fl_Tree_Item * add(Fl_Tree_Item *parent_item, const char *name);
Fl_Tree_Item *insert_above(Fl_Tree_Item *above, const char *name);
Fl_Tree_Item * insert(Fl_Tree_Item *item, const char *name, int pos);
int remove(Fl_Tree_Item *item);
void clear();
void clear_children(Fl_Tree_Item *item);

Item lookup methods
Fl_Tree_Item *find_item(const char *path);
const Fl_Tree_Item *find_item(const char *path) const;

int item_pathname(char *pathname, int pathnamelen, const Fl_Tree_Item *item) const;
Fl_Tree_Item * find_clicked(int yonly=0);
```c
Fl_Tree_Item *first(); // deprecated in ABI 10303
Fl_Tree_Item *first_visible(); // deprecated in ABI 10303
Fl_Tree_Item *next(Fl_Tree_Item *item=0);
Fl_Tree_Item *last();
Fl_Tree_Item *last_visible(); // deprecated in ABI 10303
Fl_Tree_Item *last_visible_item();
Fl_Tree_Item *next_visible_item(Fl_Tree_Item *start, int dir); // made public in 1.3.3 ABI
Fl_Tree_Item *first_selected_item();
Fl_Tree_Item *last_selected_item();
Fl_Tree_Item *next_item(Fl_Tree_Item *item, int dir=FL_Down, bool visible=false);
Fl_Tree_Item *next_selected_item(Fl_Tree_Item *item=0, int dir=FL_Down);
int get_selected_items(Fl_Tree_Item_Array &items);

// Item open/close methods
int open(Fl_Tree_Item *item, int docallback=1);
int open(const char *path, int docallback=1);
void open_toggle(Fl_Tree_Item *item, int docallback=1);
int close(Fl_Tree_Item *item, int docallback=1);
int close(const char *path, int docallback=1);
int is_open(Fl_Tree_Item *item) const;
int is_open(const char *path) const;
int is_close(Fl_Tree_Item *item) const;
int is_close(const char *path) const;

// Item selection methods
int select(Fl_Tree_Item *item, int docallback=1);
int select(const char *path, int docallback=1);
void select_toggle(Fl_Tree_Item *item, int docallback=1);
int deselect(Fl_Tree_Item *item, int docallback=1);
int deselect(const char *path, int docallback=1);
int is_selected(Fl_Tree_Item *item) const;
int is_selected(const char *path); // deprecated in ABI 10303
int is_selected_all(Fl_Tree_Item *item=0, int dir=FL_Down);
int is_selected(const char *path);
int extend_selection(Fl_Tree_Item *from, Fl_Tree_Item *to, int val);
int extend_selection_dir(Fl_Tree_Item *from, Fl_Tree_Item *to, int dir, int val);

// Item attribute related methods
Fl_Font item_labelfont() const;
void item_labelfont(Fl_Font val);
Fl_Fontsize item_labelsize() const;
void item_labelsize(Fl_Fontsize val);
Fl_Color item_labelfgcolor() const;
void item_labelfgcolor(Fl_Color val);
Fl_Color item_labelbgcolor() const;
void item_labelbgcolor(Fl_Color val);
Fl_Color connectorcolor() const;
void connectorcolor(Fl_Color val);
int marginleft() const;
void marginleft(int val);
int margintop() const;
void margintop(int val);
int marginright() const;
void marginright(int val);
int marginbottom() const;
void marginbottom(int val);
int linespacing() const;
void linespacing(int val);
int openchild_marginbottom() const;
void openchild_marginbottom(int val);
int userniconmarginleft() const;
void userniconmarginleft(int val);
int usericonmarginleft(int val);
int usericonmarginleft(int val);
int widgetmarginleft() const;
void widgetmarginleft(int val);
int connectorwidth() const;
void connectorwidth(int val);
Fl_Image* usericon() const;
void usericon(Fl_Image *val);
Fl_Image* openicon() const;
void openicon(Fl_Image *val);
Fl_Image* closeicon() const;
void closeicon(Fl_Image *val);
int showcollapse() const;
void showcollapse(int val);
int showroot() const;
```
This file contains the definitions for `Fl_Tree_Item`. 

```c
// callback related
void callback_item(Fl_Tree_Item *item);
Fl_Tree_Item *callback_item();
void callback_reason(Fl_Tree_Reason reason);
Fl_Tree_Reason callback_reason() const;

void load(class Fl_Preferences&);
```

### Classes

- **class Fl_Tree_Item**

  Tree widget item.

### Detailed Description

This file contains the definitions for `Fl_Tree_Item`.

### Fl_Tree_Item.H

Go to the documentation of this file.

```c
//
#else
#endif /*FL_TREE_ITEM_H*/
```

Generated by Doxygen
```cpp
#include <FL/Fl.H>
#include <FL/Fl_Widget.H>
#include <FL/Fl_Image.H>
#include <FL/fl_draw.H>

#include <FL/Fl_Tree_Item_Array.H>
#include <FL/Fl_Tree_Prefs.H>

// FL/Fl_Tree_Item.H
//
// Fl_Tree -- This file is part of the Fl_Tree widget for FLTK
// Copyright (C) 2009-2010 by Greg Ercolano.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
// https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
// https://www.fltk.org/bugs.php

class Fl_Tree;

class FL_EXPORT Fl_Tree_Item {
    Fl_Tree *_tree; // parent tree
    const char *_label; // label (memory managed)
    Fl_Font _labelfont; // label's font face
    Fl_Fontsize _labelsize; // label's font size
    Fl_Color _labelfgcolor; // label's fg color
    Fl_Color _labelbgcolor; // label's bg color (0xffffffff is 'transparent')

    enum Fl_Tree_Item_Flags {
        OPEN = 1«0,
        VISIBLE = 1«1,
        ACTIVE = 1«2,
        SELECTED = 1«3
    }

    unsigned short _flags; // misc flags

    int _xywh[4]; // xywh of this widget (if visible)
    int _collapse_xywh[4]; // xywh of collapse icon (if visible)

    int label_x() const { return(_label_xywh[0]); }
    int label_y() const { return(_label_xywh[1]); }
    int label_w() const { return(_label_xywh[2]); }
    int label_h() const { return(_label_xywh[3]); }

    Fl_Color drawfgcolor() const;
    Fl_Color drawbgcolor() const;

    protected:
    void _Init(const Fl_Tree_Prefs &prefs, Fl_Tree *tree);
    void show_widgets();
    void hide_widgets();
    void draw_vertical_connector(int x, int y1, int y2, const Fl_Tree_Prefs &prefs);
    void draw_horizontal_connector(int x1, int x2, int y, const Fl_Tree_Prefs &prefs);
    void recalc_tree();

    public:
    int calc_item_height(const Fl_Tree_Prefs &prefs) const;

    Fl_Tree_Item(const Fl_Tree_Prefs &prefs); // CTOR -- backwards compatible
    virtual Fl_Tree_Item(); // CTOR -- AB1 1.3.3

    Fl_Tree_Item(const Fl_Tree_Item *o); // COPY CTOR
    Fl_Tree_Item(const Fl_Tree_Item &o); // CTOR -- AB1 1.3.3+

    virtual int draw_item_content(int render);

    int x() const { return(_xywh[0]); }
    int y() const { return(_xywh[1]); }
    int w() const { return(_xywh[2]); }
    int h() const { return(_xywh[3]); }

    virtual int draw_item_content(int render);

    int *item_focus; // item focus
    int itemstate; // item state

    virtual void user_data(void *data);

    void label(const char *val);
    const char *label() const;

    const char *userdata() const;

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy
    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus, int indent = "", int render=1);

    void show_self(const char *x, int indent = "", int render=1);

    virtual void _draw(int x, int y, int w, int h);

    inline void user_data(void *data) { _userdata = data; }

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus);

    virtual void _draw(int x, int y, int w, int h);

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy

    virtual void user_data(void *data);

    virtual void draw(int x, int y, int W, Fl_Tree_Item *itemfocus); // copy

    void _draw(int x, int y, int w, int h); // copy
```
void labelfont(Fl_Font val) {
    _labelfont = val;
    recalc_tree();  // may change tree geometry
}

Fl_Font labelfont() const {
    return(_labelfont);
}

void labelfsize(Fl_Fontsize val) {
    _labelfsize = val;
    recalc_tree();  // may change tree geometry
}

Fl_Fontsize labelfsize() const {
    return(_labelfsize);
}

void labelbgcolor(Fl_Color val) {
    _labelbgcolor = val;
    labelcolor(val);
    recalc_tree();  // may change tree geometry
}

Fl_Color labelbgcolor() const {
    return(_labelbgcolor);
}

void widget(Fl_Widget *val) {
    _widget = val;
    recalc_tree();  // may change tree geometry
}

Fl_Widget *widget() const {
    return(_widget);
}

int children() const {
    return(_children.total());
}

Fl_Tree_Item *child(int index) {
    return(_children[index]);
}

const Fl_Tree_Item *child(int t) const {
    return(_children[t]);
}

int has_children() const {
    return(children());
}

int find_child(const char *name);
int find_child(Fl_Tree_Item *item);
int find_child_item(const char *name) const;
int find_child_item(char **arr) const;
int find_item(char **arr) const;

const Fl_Tree_Item *find_child_item(const char *name) const;
Fl_Tree_Item *find_child_item(char **arr);
const Fl_Tree_Item *find_item(char **arr);

// Adding items
Fl_Tree_Item *add(const Fl_Tree_Prefs &prefs, const char *new_label);
Fl_Tree_Item *add(const Fl_Tree_Prefs &prefs, char **arr);
2182 File Documentation

00251 F1_Tree_Item *next();
00252 F1_Tree_Item *next_sibling();
00253 F1_Tree_Item *prev_sibling();
00254 void update_prev_next(int index);
00255 F1_Tree_Item *next_displayed(F1_Tree_Prefs &prefs); // deprecated
00256 F1_Tree_Item *prev_displayed(F1_Tree_Prefs &prefs); // deprecated
00257 F1_Tree_Item *next_visible(F1_Tree_Prefs &prefs);
00258 F1_Tree_Item *prev_visible(F1_Tree_Prefs &prefs);
00259
00261 F1_Tree_Item *parent() {
00262 return(_parent);
00263 }
00265 const F1_Tree_Item *parent() const {
00266 return(_parent);
00267 }
00271 void parent(F1_Tree_Item *val) {
00272 _parent = val;
00273 }
00274 const F1_Tree_Prefs& prefs() const;
00277 const F1_Tree *tree() const {
00278 return(_tree);
00279 }
00282 F1_Tree *tree() {
00283 return(_tree);
00284 }
00286 // State
00288 void open();
00289 void close();
00291 int is_open() const {
00292 return(is_flag(OPEN));
00293 }
00295 int is_close() const {
00296 return(is_flag(OPEN)?0:1);
00297 }
00299 void open_toggle() {
00300 is_open()?close():open(); // handles calling recalc_tree()
00301 }
00305 void select(int val=1) {
00306 set_flag(SELECTED, val);
00307 }
00309 void select_toggle() {
00310 if { is_selected() } {
00311 deselect(); // deselect if selected
00312 } else {
00313 select(); // select if deselected
00314 }
00320 int select_all() {
00321 int count = 0;
00322 if { ! is_selected() } {
00323 select();
00324 ++count;
00325 }
00326 for { int t=0; t<children(); t++ } {
00327 count += child(t)->select_all();
00328 }
00329 return(count);
00330 }
00332 void deselect() {
00333 set_flag(SELECTED, 0);
00334 }
00339 int deselect_all() {
00340 int count = 0;
00342 if { is_selected() } {
00343 deselect();
00349 }
00344 }
00354 for { int t=0; t<children(); t++ } {
00356 count += child(t)->deselect_all();
00357 }
00358 return(count);
00359 }
00365 char is_selected() const {
00366 return(is_flag(SELECTED));
00367 }
00368 void activate(int val=1) {
00369 set_flag(ACTIVE, val);
00371 if { _widget && val != (int)_widget->active() } {
00373 if { val } {
00375 _widget->activate();
00376 } else {
00377 _widget->deactivate();
00378 }
00380 _widget->redraw();
00381 }
00382 }
00387 void deactivate() {
This file defines a class that manages an array of Fl_Tree_Item pointers.

#include <FL/Fl.H>
#include "Fl_Export.H"

Classes

- class Fl_Tree_Item_Array
  
  Manages an array of Fl_Tree_Item pointers.

13.164.1 Detailed Description

This file defines a class that manages an array of Fl_Tree_Item pointers.
Go to the documentation of this file.

```c
//
#ifndef _FL_TREE_ITEM_ARRAY_H
#define _FL_TREE_ITEM_ARRAY_H

#include <FL/Fl.H>
#include "Fl_Export.H"

class FL_EXPORT Fl_Tree_Item; // forward decl must precede first doxygen comment block
// or doxygen will not document our class..

// FL/Fl_Tree_Item_Array.H
//
// Fl_Tree -- This file is part of the Fl_Tree widget for FLTK
// Copyright (C) 2009-2010 by Greg Ercolano.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
// https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
// https://www.fltk.org/bugs.php

class FL_EXPORT Fl_Tree_Item_Array { 

    Fl_Tree_Item **_items; // items array
    int _total; // Items in array
    int _size; // Items allocated, for array
    int _chunksize; // Items to enlarge mem allocation
    enum {
        MANAGE_ITEM = 1
    }
    char _flags; // flags to control behavior

    void enlarge(int count);

public:

    Fl_Tree_Item_Array(int new_chunksize = 10); // CTOR
    ~Fl_Tree_Item_Array(); // DTOR
    Fl_Tree_Item_Array(const Fl_Tree_Item_Array *o); // COPY CTOR

    Fl_Tree_Item *operator[](int i) {
        return(_items[i]);
    }
    const Fl_Tree_Item *operator[](int i) const {
        return(_items[i]);
    }

    int total() const {
        return(_total);
    }

    void swap(int ax, int bx);
    int move(int to, int from);
    int deparent(int pos);
    int reparent(Fl_Tree_Item *item, Fl_Tree_Item *newparent, int pos);
    void clear();
    void add(Fl_Tree_Item *val);
    void insert(int pos, Fl_Tree_Item *new_item);
    void replace(int pos, Fl_Tree_Item *new_item);
    void remove(int index);
    void remove(Fl_Tree_Item *item);
    void manage_item_destroy(int val) {
        if ( val ) _flags |= MANAGE_ITEM; else _flags &= ~MANAGE_ITEM;
    }
    int manage_item_destroy() const {
        return _flags & MANAGE_ITEM ? 1 : 0;
    }

} // class Fl_Tree_Item_Array

#endif /*_FL_TREE_ITEM_ARRAY_H*/
```

This file contains the definitions for Fl_Tree's preferences.

```
#include <FL/Fl.H>
```

### 13.165 Fl_Tree_Item_Array.H

This file is part of the Fl_Tree widget for FLTK.

### 13.166 Fl_Tree_Prefs.H File Reference

This file contains the definitions for Fl_Tree's preferences.

---

Generated by Doxygen
Classes

- class Fl_Tree_Prefs
  
  Tree widget's preferences.

TypeDefs

- typedef void() Fl_Tree_Item_Draw_Callback(Fl_Tree_Item *, void *)

Enumerations

- enum Fl_Tree_Connector { FL_TREE_CONNECTOR_NONE =0, FL_TREE_CONNECTOR_DOTTED =1, FL_TREE_CONNECTOR_SOLID =2 }

  Defines the style of connection lines between items.

- enum Fl_Tree_Item_Draw_Mode { FL_TREE_ITEM_DRAW_DEFAULT =0, FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET =1, FL_TREE_ITEM_HEIGHT_FROM_WIDGET =2 }

  Bit flags that control how item's labels and widget()s are drawn in the tree via item_draw_mode().

- enum Fl_Tree_Item_Reselect_Mode { FL_TREE_SELECTABLE_ONCE =0, FL_TREE_SELECTABLE_ALWAYS }

  Defines the ways an item can be (re) selected via item_reselect_mode().

- enum Fl_Tree_Select { FL_TREE_SELECT_NONE =0, FL_TREE_SELECT_SINGLE =1, FL_TREE_SELECT_MULTI =2, FL_TREE_SELECT_SINGLE_DRAGGABLE =3 }

  Tree selection style.

- enum Fl_Tree_Sort { FL_TREE_SORT_NONE =0, FL_TREE_SORT_ASCENDING =1, FL_TREE_SORT_DESCENDING =2 }

  Sort order options for items added to the tree.

13.166.1 Detailed Description

This file contains the definitions for Fl_Tree's preferences.

```
Fl_Tree_Prefs

Fl_Tree

Fl_Tree_Item
```

13.166.2 Enumeration Type Documentation

13.166.2.1 Fl_Tree_Connector

```
enum Fl_Tree_Connector

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_CONNECTOR_NONE</td>
<td>Use no lines connecting items.</td>
</tr>
<tr>
<td>FL_TREE_CONNECTOR_DOTTED</td>
<td>Use dotted lines connecting items (default)</td>
</tr>
<tr>
<td>FL_TREE_CONNECTOR_SOLID</td>
<td>Use solid lines connecting items.</td>
</tr>
</tbody>
</table>
```

13.166.2.2 Fl_Tree_Item_Draw_Mode

```
enum Fl_Tree_Item_Draw_Mode

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_ITEM_DRAW_DEFAULT</td>
<td>If widget() defined, draw in place of label, and widget() tracks item height (default)</td>
</tr>
</tbody>
</table>
```
### FL_Tree_Item_Reselect_Mode

```c
enum FL_Tree_Item_Reselect_Mode
```

Defines the ways an item can be (re) selected via item_reselect_mode().

- **FL_TREE_SELECTABLE_ONCE**: Item can only be selected once (default).
- **FL_TREE_SELECTABLE_ALWAYS**: Enables FL_TREE_REASON_RESELECTED events for callbacks.

### FL_Tree_Select

```c
enum FL_Tree_Select
```

Tree selection style.

- **FL_TREE_SELECT_NONE**: Nothing selected when items are clicked.
- **FL_TREE_SELECT_SINGLE**: Single item selected when item is clicked (default).
- **FL_TREE_SELECT_MULTI**: Multiple items can be selected by clicking with SHIFT, CTRL or mouse drags.
- **FL_TREE_SELECT_SINGLE_DRAGGABLE**: Single items may be selected, and they may be reordered by mouse drag.

### FL_Tree_Sort

```c
enum FL_Tree_Sort
```

Sort order options for items added to the tree.

- **FL_TREE_SORT_NONE**: No sorting; items are added in the order defined (default).
- **FL_TREE_SORT_ASCENDING**: Add items in ascending sort order.
- **FL_TREE_SORT_DESCENDING**: Add items in descending sort order.

---

Go to the documentation of this file.

```c
00001 //
00002
00003 ifndef FL_TREE_PREFS_H
00004 #define FL_TREE_PREFS_H
00005
00006 #include <FL/Fl.H> // needed for ABI version features (via Enumerations.H)
00007 00008 #endif // FL_Tree_PREFS_H
00009 00010 // FL/Fl_Tree_Prefs.H
00011 //
00012 // FL_TreePrefs -- This file is part of the FL_Tree widget for FLTK
00013 // Copyright (C) 2009-2010 by Greg Ercolano.
00014 //
```
enum Fl_Tree_Sort {
    FL_TREE_SORT_NONE = 0,
    FL_TREE_SORT_ASCENDING = 1,
    FL_TREE_SORT_DESCENDING = 2
};

enum Fl_Tree_Connector {
    FL_TREE_CONNECTOR_NONE = 0,
    FL_TREE_CONNECTOR_DOTTED = 1,
    FL_TREE_CONNECTOR_SOLID = 2
};

enum Fl_Tree_Select {
    FL_TREE_SELECT_NONE = 0,
    FL_TREE_SELECT_SINGLE = 1,
    FL_TREE_SELECT_MULTI = 2,
    FL_TREE_SELECT_SINGLE_DRAGGABLE = 3
};

enum Fl_Tree_Item_Reselect_Mode {
    FL_TREE_SELECTABLE_ONCE = 0,
    FL_TREE_SELECTABLE_ALWAYS
};

enum Fl_Tree_Item_Draw_Mode {
    FL_TREE_ITEM_DRAW_DEFAULT = 0,
    FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET = 1,
    FL_TREE_ITEM_HEIGHT_FROM_WIDGET = 2
};

class Fl_Tree_Item;

typedef void (Fl_Tree_Item_Draw_Callback)(Fl_Tree_Item*, void*);

class FL_EXPORT Fl_Tree_Prefs {
    Fl_Font _labelfont; // label's font face
    Fl_Fontsize _labelsize; // label's font size
    int _margintop; // --
    int _marginleft; // |-- tree's controllable margins
    int _marginbottom; // --
    int _openchild_marginbottom; // extra space below an open child tree
    int _usericonmarginleft; // space to left of user icon (if any)
    int _labelmarginleft; // |-- label's margin left
    int _widgetmarginleft; // space to left of widget
    int _connectorwidth; // connector width (right of open/close icon)
    int _linespacing; // vertical space between lines

    // Colors
    Fl_Color _labelfgcolor; // label's foreground color
    Fl_Color _labelbgcolor; // label's background color
    int _usericonmarginleft; // space to left of user icon (if any)
    int _labelmarginleft; // |-- label's margin left
    Fl_Draw_Callback _itemdrawcallback; // callback to handle drawing items (0=none)
    void* _itemdrawuserdata; // data for drawing items (0=none)

public:
    Fl_Tree_Prefs();
    ~Fl_Tree_Prefs();

    // Labels
    inline Fl_Font item_labelfont() const { return(_labelfont); }  
    inline void item_labelfont(Fl_Font val) { _labelfont = val; }  
    inline Fl_Fontsize item_labelsize() const { return(_labelsize); } 
    inline void item_labelsize(Fl_Fontsize val) { _labelsize = val; }
inline Fl_Color item_labelfgcolor() const { return(_labelfgcolor); }
inline void item_labelfgcolor(Fl_Color val) { _labelfgcolor = val; }
inline Fl_Color item_labelbgcolor() const {
    return _labelbgcolor;
}
inline void item_labelbgcolor(Fl_Color val) {
    _labelbgcolor = val;
}
inline Fl_Color labelfgcolor() const { return(_labelfgcolor); }
inline void labelfgcolor(Fl_Color val) { _labelfgcolor = val; }
inline Fl_Font labelfont() const { return(_labelfont); }
inline void labelfont(Fl_Font val) { _labelfont = val; }
inline Fl_Fontsize labelsize() const { return(_labelsize); }
inline void labelsize(Fl_Fontsize val) { _labelsize = val; }
inline Fl_Color labelbgcolor() const { return(item_labelbgcolor()); }
inline void labelbgcolor(Fl_Color val) { item_labelbgcolor(val); }

// Obsolete names - for 1.3.0 backwards compat
inline Fl_Font labelfont() const { return(_labelfont); }
inline void labelfont(Fl_Font val) { _labelfont = val; }
inline Fl_Fontsize labelsize() const { return(_labelsize); }
inline void labelsize(Fl_Fontsize val) { _labelsize = val; }
inline Fl_Color labelfgcolor() const { return(_labelfgcolor); }
inline void labelfgcolor(Fl_Color val) { _labelfgcolor = val; }
inline Fl_Color labelbgcolor() const { return(item_labelbgcolor()); }
inline void labelbgcolor(Fl_Color val) { item_labelbgcolor(val); }

// Margins
inline int marginleft() const { return(_marginleft); }
inline void marginleft(int val) {
    _marginleft = val;
}
inline int margintop() const { return(_margintop); }
inline void margintop(int val) {
    _margintop = val;
}
inline int marginbottom() const { return(_marginbottom); }
inline void marginbottom(int val) {
    _marginbottom = val;
}
inline int openchild_marginbottom() const {
    return(_openchild_marginbottom);
}
inline void openchild_marginbottom(int val) {
    _openchild_marginbottom = val;
}
inline int usericonmarginleft() const { return(_usericonmarginleft); }
inline void usericonmarginleft(int val) {
    _usericonmarginleft = val;
}
inline int labellinearleft() const { return(_labellinearleft); }
inline void labellinearleft(int val) {
    _labellinearleft = val;
}
inline int widgetmarginleft() const { return(_widgetmarginleft); }
inline void widgetmarginleft(int val) {
    _widgetmarginleft = val;
}
inline int linespacing() const { return(_linespacing); }
inline void linespacing(int val) {
    _linespacing = val;
}

// Colors and Styles
inline Fl_Color connectorcolor() const { return(_connectorcolor); }
inline void connectorcolor(Fl_Color val) { _connectorcolor = val; }
inline Fl_Tree_Connector connectorstyle() const { return(_connectorstyle); }
inline void connectorstyle(Fl_Tree_Connector val) { _connectorstyle = val; }
inline void connectorstyle(int val) { _connectorstyle = Fl_Tree_Connector(val); }
inline int connectorwidth() const { return(_connectorwidth); }

// Generated by Doxygen
inline void connectorwidth(int val) {
  _connectorwidth = val;
}

// Icons
inline Fl_Image *openicon() const {
  return(_openimage);
}
void openicon(Fl_Image *val);
inline Fl_Image *closeicon() const {
  return(_closeimage);
}
void closeicon(Fl_Image *val);
inline Fl_Image *usericon() const {
  return(_userimage);
}
inline void usericon(Fl_Image *val) {
  _userimage = val;
}
// Update deactivated version of icon..
if (!_userimage) delete _userimage;
if (!_userimage) {
  _userimage = _userimage->copy();
  _userimage->inactive();
} else {
  _userimage = 0;
}

inline Fl_Image *opendeicon() const {
  return _opendeimage;
}
inline Fl_Image *closedeicon() const {
  return _closedeimage;
}
inline Fl_Image *userdeicon() const {
  return _userdeimage;
}

// Options
inline char showcollapse() const {
  return(_showcollapse);
}
inline void showcollapse(int val) {
  _showcollapse = val;
}
inline Fl_Tree_Sort sortorder() const {
  return(_sortorder);
}
inline void sortorder(Fl_Tree_Sort val) {
  _sortorder = val;
}
inline Fl_Boxtype selectbox() const {
  return(_selectbox);
}
inline void selectbox(Fl_Boxtype val) {
  _selectbox = val;
}
inline void selectbox(Fl_Boxtype val) {
  _selectbox = val;
}
inline void showroot(int val) {
  _showroot = char(val);
}
inline Fl_Tree_Select selectmode() const {
  return(_selectmode);
}
inline void selectmode(Fl_Tree_Select val) {
  _selectmode = val;
}
```c
void * item_draw_user_data() const {
    return(_itemdrawuserdata);
}
void do_item_draw_callback(Fl_Tree_Item *o) const {
    _itemdrawcallback(o, _itemdrawuserdata);
}
```

### 13.168 fl_types.h File Reference

This file contains simple "C"-style type definitions.

```c
#include "fl_attr.h"
```

#### Typedefs

##### Miscellaneous

- `typedef unsigned int Fl_Shortcut`
  
  16-bit Unicode character + 8-bit indicator for keyboard flags.

- `typedef unsigned char uchar`

- `typedef unsigned long ulong`

#### 13.168.1 Detailed Description

This file contains simple "C"-style type definitions.

#### 13.168.2 Typedef Documentation

##### 13.168.2.1 Fl_Shortcut

`typedef unsigned int Fl_Shortcut`

16-bit Unicode character + 8-bit indicator for keyboard flags.

**Note**

This should be 24-bit Unicode character + 8-bit indicator for keyboard flags. The upper 8 bits are currently unused but reserved.

Due to compatibility issues this type and all FLTK shortcuts can only be used with 16-bit Unicode characters (U+0000 .. U+FFFF) and not with the full range of unicode characters (U+0000 .. U+10FFFF). This is caused by the bit flags FL_SHIFT, FL_CTRL, FL_ALT, and FL_META being all in the range 0x010000 .. 0x400000.

**Todo**

Discuss and decide whether we can "shift" these special keyboard flags to the upper byte to enable full 21-bit Unicode characters (U+0000 .. U+10FFFF) plus the keyboard indicator bits as this was originally intended. This would be possible if we could rely on all programs being coded with symbolic names and not hard coded bit values.

#### 13.169 fl_types.h

Go to the documentation of this file.
13.170 fl_utf8.h File Reference

header for Unicode and UTF-8 character handling
#include "FL_Export.H"
#include "fl_types.h"
#include <stdio.h>
#include <sys/stat.h>

Functions

- int fl_access (const char *f, int mode)
  Cross-platform function to test a files access() with a UTF-8 encoded name or value.

- int fl_chdir (const char *path)
  Cross-platform function to change the current working directory, given as a UTF-8 encoded string.

- int fl_chmod (const char *f, int mode)
  Cross-platform function to set a files mode() with a UTF-8 encoded name or value.

- int fl_close_fd (int fd)
  Cross-platform function to close a file descriptor.

- int fl_execvp (const char *file, char **const argv)
  Cross-platform function to open files with a UTF-8 encoded name.

- FILE * fl_fopen (const char *f, const char *mode)
  Cross-platform function to open files with a UTF-8 encoded name or value.

- char * fl_getcwd (char *buf, int len)
  Cross-platform function to get the current working directory as a UTF-8 encoded value.

- char * fl_getenv (const char *v)
  Cross-platform function to get environment variables with a UTF-8 encoded name or value.

- char fl_make_path (const char *path)
  Cross-platform function to recursively create a path in the file system.

- void fl_make_path_for_file (const char *path)
  Cross-platform function to create a path for the file in the file system.

- int fl_mkdir (const char *f, int mode)
  Cross-platform function to create a directory with a UTF-8 encoded name.

- unsigned int fl_nonspacing (unsigned int ucs)
  Returns true if the Unicode character ucs is non-spacing.

- int fl_open (const char *fname, int oflags,...)
  Cross-platform function to open files with a UTF-8 encoded name.

- int fl_open_ext (const char *fname, int binary, int oflags,...)
  Cross-platform function to open files with a UTF-8 encoded name.
• int fl_putenv (const char *var)
  Cross-platform function to write environment variables with a UTF-8 encoded name or value.
• int fl_rename (const char *f, const char *n)
  Cross-platform function to rename a filesystem object using UTF-8 encoded names.
• int fl_rmdir (const char *f)
  Cross-platform function to remove a directory with a UTF-8 encoded name.
• int fl_stat (const char *f, struct stat *b)
  Cross-platform function to stat() a file using a UTF-8 encoded name or value.
• int fl_system (const char *cmd)
  Cross-platform function to run a system command with a UTF-8 encoded string.
• int fl_tolower (unsigned int ucs)
  Returns the Unicode lower case value of ucs.
• int fl_toupper (unsigned int ucs)
  Returns the Unicode upper case value of ucs.
• unsigned fl_ucs_to_Utf16 (const unsigned ucs, unsigned short *dst, const unsigned dstlen)
  Convert a single 32-bit Unicode codepoint into an array of 16-bit characters.
• int fl_unlink (const char *fname)
  Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename.
• char * fl_utf2mbcs (const char *s)
  Converts UTF-8 string s to a local multi-byte character string.
• const char * fl_utf8back (const char *p, const char *start, const char *end)
  Move p backward until it points to the start of a UTF-8 character.
• int fl_utf8bytes (unsigned ucs)
  Return the number of bytes needed to encode the given UCS4 character in UTF-8.
• unsigned fl_utf8decode (const char *p, const char *end, int *len)
  Decode a single UTF-8 encoded character starting at p.
• int fl_utf8encode (unsigned ucs, char *buf)
  Write the UTF-8 encoding of ucs into buf and return the number of bytes written.
• unsigned fl_utf8from_mb (char *dst, unsigned dstlen, const char *src, unsigned srclen)
  Convert a filename from the locale-specific multibyte encoding used by Windows to UTF-8 as used by FLTK.
• unsigned fl_utf8froma (const char *dst, unsigned dstlen, const char *src, unsigned srclen)
  Convert an ISO-8859-1 (ie normal c-string) byte stream to UTF-8.
• unsigned fl_utf8fromwc (char *dst, unsigned dstlen, const wchar_t *src, unsigned srclen)
  Turn "wide characters" as returned by some system calls (especially on Windows) into UTF-8.
• const char * fl_utf8fwd (const char *p, const char *start, const char *end)
  Move p forward until it points to the start of a UTF-8 character.
• int fl_utf8len (char c)
  Returns the byte length of the UTF-8 sequence with first byte c, or -1 if c is not valid.
• int fl_utf8len1 (char c)
  Returns the byte length of the UTF-8 sequence with first byte c, or 1 if c is not valid.
• int fl_utf8locale ()
  Return true if the "locale" seems to indicate that UTF-8 encoding is used.
• int fl_utf8strlen (const char *text, int len)
  Return the length in bytes of a UTF-8 string.
• int fl_utf8test (const char *src, unsigned srclen)
  Examines the first srclen bytes in src and returns a verdict on whether it is UTF-8 or not.
• unsigned fl_utf8to_mb (const char *src, unsigned srclen, char *dst, unsigned dstlen)
  Convert the UTF-8 used by FLTK to the locale-specific encoding used for filenames (and sometimes used for data in files).
• unsigned fl_utf8toa (const char *src, unsigned srclen, char *dst, unsigned dstlen)
  Convert a UTF-8 sequence into an array of 1-byte characters.
• unsigned fl_utf8toUtf16 (const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen)
   *Convert a UTF-8 sequence into an array of 16-bit characters.*

• unsigned fl_utf8towc (const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen)
   *Converts a UTF-8 string into a wide character string.*

• int fl_utf_nb_char (const unsigned char *buf, int len)
   *Returns the number of Unicode chars in the UTF-8 string.*

• int fl_utf_strcasecmp (const char *s1, const char *s2)
   *UTF-8 aware strcasecmp - converts to Unicode and tests.*

• int fl_utf_strncasecmp (const char *s1, const char *s2, int n)
   *UTF-8 aware strncasecmp - converts to lower case Unicode and tests.*

• int fl_utf_tolower (const unsigned char *str, int len, char *buf)
   *Converts the string str to its lower case equivalent into buf.*

• int fl_utf_toupper (const unsigned char *str, int len, char *buf)
   *Converts the string str to its upper case equivalent into buf.*

• int fl_wcwidth (const char *src)
   *extended wrapper around fl_wcwidth_(unsigned int ucs) function.*

• int fl_wcwidth_ (unsigned int ucs)
   *Wrapper to adapt Markus Kuhn's implementation of wcwidth() for FLTK.*

13.170.1 Detailed Description

header for Unicode and UTF-8 character handling

13.171 fl_utf8.h

Go to the documentation of this file.
/* OD: returns the byte length of a UTF-8 text */
FL_EXPORT int fl_utf8strlen(const char *text, int len);

/* OD: returns the number of Unicode chars in the UTF-8 string */
FL_EXPORT int fl_utf_nb_char(const unsigned char *buf, int len);

/* F2: Convert the next UTF-8 char-sequence into a Unicode value (and say how many bytes were used) */
FL_EXPORT unsigned fl_utf8decode(const char *p, const char *end, int *len);

/* F2: Encode a Unicode value into a UTF-8 sequence, return the number of bytes used */
FL_EXPORT int fl_utf8encode(unsigned ucs, char *buf);

/* F2: Move forward to the next valid UTF-8 sequence start between start and end */
FL_EXPORT const char* fl_utf8fwd(const char* p, const char* start, const char* end);

/* F2: Move backward to the previous valid UTF-8 sequence start */
FL_EXPORT const char* fl_utf8back(const char* p, const char* start, const char* end);

/* XX: Convert a single 32-bit Unicode value into UTF16 */
FL_EXPORT unsigned fl_ucs_to_Utf16(const unsigned ucs, unsigned short *dst, const unsigned dstlen);

/* F2: Convert a UTF-8 string into UTF16 */
FL_EXPORT unsigned fl_utf8toUtf16(const char* src, unsigned srclen, unsigned short* dst, unsigned dstlen);

/* F2: Convert a UTF-8 string into a wide character string - makes UTF16 on win32, "UCS4" elsewhere */
FL_EXPORT unsigned fl_utf8towc(const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen);

/* F2: Convert a wide character string to UTF-8 - takes in UTF16 on win32, "UCS4" elsewhere */
FL_EXPORT unsigned fl_utf8fromwc(char *src, unsigned srclen, char *dst, unsigned dstlen);

/* F2: Convert a UTF-8 string into ASCII, eliding untranslatable glyphs */
FL_EXPORT unsigned fl_utf8toa (const char *src, unsigned srclen, char *dst, unsigned dstlen);

/* F2: Convert 8859-1 string to UTF-8 */
FL_EXPORT unsigned fl_utf8froma (char *dst, unsigned dstlen, const char *src, unsigned srclen);

/* F2: Returns true if the current O/S locale is UTF-8 */
FL_EXPORT int fl_utf8locale(void);

/* F2: Examine the first len characters of src, to determine if the input text is UTF-8 or not */
FL_EXPORT int fl_utf8test (const char *src, unsigned len);

/* XX: return width of "raw" ucs character in columns. *for internal use only*/
FL_EXPORT int fl_wcwidth_(unsigned int ucs);

/* XX: return width of UTF-8 character string in columns. *NOTE: this may also do C1 control character (0x80 to 0x9f) to CP1252 mapping, */
FL_EXPORT int fl_wcwidth (const char *src);

/* OD: UTF-8 aware strncasecmp - converts to lower case Unicode and tests */
FL_EXPORT int fl_utf_strncasecmp(const char *s1, const char *s2, int n);

/* OD: UTF-8 aware strcasecmp - converts to Unicode and tests */
FL_EXPORT int fl_utf_strcasecmp(const char *s1, const char *s2);

Generated by Doxygen
/* OD: return the Unicode lower case value of ucs */
FL_EXPORT int fl_tolower(unsigned int ucs);

/* OD: return the Unicode upper case value of ucs */
FL_EXPORT int fl_toupper(unsigned int ucs);

/* OD: converts the UTF-8 string to the lower case equivalent */
FL_EXPORT int fl_utf_tolower(const unsigned char *str, int len, char *buf);

/* OD: converts the UTF-8 string to the upper case equivalent */
FL_EXPORT int fl_utf_toupper(const unsigned char *str, int len, char *buf);

/* OD: Portable UTF-8 aware chmod wrapper */
FL_EXPORT int fl_chmod(const char* f, int mode);

/* OD: Portable UTF-8 aware access wrapper */
FL_EXPORT int fl_access(const char* f, int mode);

/* OD: Portable UTF-8 aware stat wrapper */
FL_EXPORT int fl_stat(const char *path, struct stat *buffer);

/* Portable UTF-8 aware chdir wrapper */
FL_EXPORT int fl_chdir(const char *path);

/* OD: Portable UTF-8 aware fopen wrapper */
FL_EXPORT FILE *fl_fopen(const char *f, const char *mode);

/* OD: Portable UTF-8 aware system wrapper */
FL_EXPORT int fl_system(const char* f);

/* OD: Portable UTF-8 aware execvp wrapper */
FL_EXPORT int fl_execvp(const char *file, char *const *argv);

/* OD: Portable UTF-8 aware open wrapper */
FL_EXPORT int fl_open(const char *fname, int oflags, ...);

/* OD: Portable UTF-8 aware unlink wrapper */
FL_EXPORT int fl_unlink(const char *fname);

/* OD: Portable UTF-8 aware rename wrapper */
FL_EXPORT int fl_rename(const char* f, const char *t);

/* OD: Given a full pathname, this will create the directory path needed to hold the file named */
FL_EXPORT void fl_make_path_for_file( const char *path );

/* OD: recursively create a path in the file system */
FL_EXPORT char fl_make_path( const char *path );

#ifndef __cplusplus
}
#endif /* __cplusplus */
The Fl_Valuator widget

```c
/*
 * file Fl_Valuator widget . */

#ifndef Fl_Valuator_H
#define Fl_Valuator_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

// shared type() values for classes that work in both directions:
define FL_VERTICAL 0
define FL_HORIZONTAL 1

class FL_EXPORT Fl_Valuator : public Fl_Widget {

protected:
  int horizontal() const {return type()& FL_HORIZONTAL;}
  Fl_Valuator(int X, int Y, int W, int H, const char * L);

  double previous_value() const {return previous_value_ ;}
  void handle_push() {previous_value_ = value_ ;}
  double softclamp(double);
  void handle_drag(double newvalue);
  void handle_release(); // use drag() value
  virtual void value_damage(); // cause damage() due to value() changing
  void set_value(double v) {value_ = v ;}

public:
  void bounds(double a, double b) {min=a; max=b;}
  double minimum() const {return min;}
  void minimum(double a) {min = a;}
  double maximum() const {return max;}
  void maximum(double a) {max = a;}
  void range(double a, double b) {min = a; max = b;}
  void step(int a) {A = a; B = 1;}
  void step(double a, int b) {A = a; B = b;}
  void step(double s);
  double step() const {return A/B;}
  void precision(int digits);
  double value() const {return value_ ;}
  int value(double);
  virtual int format(char*);
  double round(double); // round to nearest multiple of step
  double clamp(double); // keep in range
  double increment(double, int); // add n*step to value

  protected:
  int horizontal() const {return type()& FL_HORIZONTAL;}
  Fl_Valuator(int X, int Y, int W, int H, const char * L);
  double previous_value() const {return previous_value_ ;}
  void handle_push() {previous_value_ = value_ ;}
  double softclamp(double);
  void handle_drag(double newvalue);
  void handle_release(); // use drag() value
  virtual void value_damage(); // cause damage() due to value() changing
  void set_value(double v) {value_ = v ;}
}
#endif
```

---

**13.173 Fl_Value_Input.H**

```c
#ifndef Fl_Value_Input_H
#define Fl_Value_Input_H

#ifndef Fl_Value_H
#include "Fl_Value.H"
#endif

class FL_EXPORT Fl_Value_Input : public Fl_Value {

public:
  // Value input header file for the Fast Light Tool Kit (FLTK).

private:
  Fl_Value_Input();

protected:
  int horizontal() const {return type()& FL_HORIZONTAL;}
  Fl_Valuator(int X, int Y, int W, int H, const char * L);
  double previous_value() const {return previous_value_ ;}
  void handle_push() {previous_value_ = value_ ;}
  double softclamp(double);
  void handle_drag(double newvalue);
  void handle_release(); // use drag() value
  virtual void value_damage(); // cause damage() due to value() changing
  void set_value(double v) {value_ = v ;}

  protected:
  int horizontal() const {return type()& FL_HORIZONTAL;}
  Fl_Valuator(int X, int Y, int W, int H, const char * L);
  double previous_value() const {return previous_value_ ;}
  void handle_push() {previous_value_ = value_ ;}
  double softclamp(double);
  void handle_drag(double newvalue);
  void handle_release(); // use drag() value
  virtual void value_damage(); // cause damage() due to value() changing
  void set_value(double v) {value_ = v ;}
}
#endif
```
/* Fl_Value_Input.h */

#ifndef Fl_Value_Input_H
#define Fl_Value_Input_H

#include "Fl_Valuator.H"
#include "Fl_Input.H"

class FL_EXPORT Fl_Value_Input : public Fl_Valuator {
    Fl_Input input;
    char soft_;  
    static void input_cb(Fl_Widget *, void*);
    void value_damage() FL_OVERRIDE;  

public:
    Fl_Value_Input(int x, int y, int w, int h, const char *l=0);
    ~Fl_Value_Input();
    void soft(char s) {soft_ = s;}
    char soft() const {return soft_;}
    int shortcut() const {return input.shortcut();}
    void shortcut(int s) {input.shortcut(s);}
    Fl_Font textfont() const {return input.textfont();}
    void textfont(Fl_Font s) {input.textfont(s);}
    Fl_Fontsize textsize() const {return input.textsize();}
    void textsize(Fl_Fontsize s) {input.textsize(s);}
    Fl_Color textcolor() const {return input.textcolor();}
    void textcolor(Fl_Color n) {input.textcolor(n);}
    Fl_Color cursor_color() const {return input.cursor_color();}
    void cursor_color(Fl_Color n) {input.cursor_color(n);}

protected:
    int handle(int) FL_OVERRIDE;
    void draw() FL_OVERRIDE;
    void resize(int, int, int, int) FL_OVERRIDE;
};

#endif
public:

int handle(int) FL_OVERRIDE;

Fl_Value_Output(int x, int y, int w, int h, const char *l = 0);

void soft(uchar s) {soft_ = s;}

uchar soft() const {return soft_;

Fl_Font textfont() const {return textfont_;}

void textfont(Fl_Font s) {textfont_ = s;

Fl_Fontsize textsize() const {return textsize_;}

void textsize(Fl_Fontsize s) {textsize_ = s;

Fl_Color textcolor() const {return textcolor_;}

void textcolor(Fl_Color s) {textcolor_ = s;

protected:

void value_width(int s) {
if (s > w() - 10)
s = w() - 10;
if (s < 10)
s = 10;
value_width_ = (short)s;

int value_width() const { return (value_width_); }

void value_height(int s) {
if (s > h() - 10)
s = h() - 10;
if (s < 10)
s = 10;
value_height_ = (short)s;

13.175 Fl_Value_Slider.H

class FL_EXPORT Fl_Value_Slider : public Fl_Slider {
Fl_Font textfont_;
Fl_Fontsize textsize_;
Fl_Color textcolor_;
short value_width_;
short value_height_;
13.176  Fl_Widget.H File Reference

Fl_Widget and Fl_Label classes.
#include "Fl.H"

Classes

- class Fl_Callback_User_Data
  A class prototype that allows for additional data in callbacks.
- struct Fl_Label
  This struct stores all information for a text or mixed graphics label.
- class Fl_Widget
  Fl_Widget is the base class for all widgets in FLTK.

Macros

- #define FL_RESERVED_TYPE 100
  Reserved type numbers (necessary for my cheapo RTTI) start here.

Typedefs

- typedef void() Fl_Callback(Fl_Widget *, void *)
  Default callback type definition for all fltk widgets (by far the most used)
- typedef void() Fl_Callback0(Fl_Widget *)
  One parameter callback type definition passing only the widget.
- typedef void() Fl_Callback1(Fl_Widget *, long)
  Callback type definition passing the widget and a long data value.
- typedef Fl_Callback * Fl_Callback_p
  Default callback type pointer definition for all fltk widgets.

13.176.1  Detailed Description

Fl_Widget and Fl_Label classes.

13.176.2  Macro Definition Documentation

13.176.2.1  FL_RESERVED_TYPE

#define FL_RESERVED_TYPE 100
Reserved type numbers (necessary for my cheapo RTTI) start here.
Grep the header files for "RESERVED_TYPE" to find the next available number.

13.177  Fl_Widget.H

Go to the documentation of this file.
#ifndef Fl_Widget_H
#define Fl_Widget_H

#include "Fl.H"

class Fl_Widget;
class Fl_Window;
class Fl_Group;
class Fl_Image;

typedef void (Fl_Callback )(Fl_Widget*, void*);
typedef Fl_Callback* Fl_Callback_p; // needed for BORLAND
typedef void (Fl_Callback0)(Fl_Widget*);
typedef void (Fl_Callback1)(Fl_Widget*, long);

struct FL_EXPORT Fl_Label {
    const char * value;
    Fl_Image * image;
    Fl_Image * deimage;
    Fl_Font font;
    Fl_Fontsize size;
    Fl_Color color;
    Fl_Align align_;
    uchar type;

    void draw(int,int,int,int, Fl_Align) const ;
    void measure(int &w, int &h) const ;
};

class Fl_Callback_User_Data {
    public:
        virtual ~Fl_Callback_User_Data() { }
};

class FL_EXPORT Fl_Widget {
    friend class Fl_Group;

    Fl_Group * parent_;
    Fl_Callback * callback_;
    void * user_data_
    int x_,y_,w_,h_;
    Fl_Label label_
    unsigned int flags_
    Fl_Color color_
    Fl_Color color2_
    uchar type_
    uchar damage_
    uchar box_
    uchar when_
    const char * tooltip_

    Fl_Widget(const Fl_Widget &);
    Fl_Widget& operator=(const Fl_Widget &);

    protected:

    Fl_Widget(int x, int y, int w, int h, const char *label=0L);

    void x(int v) {x_ = v;}
    void y(int v) {y_ = v;}
    void w(int v) {w_ = v;}
    void h(int v) {h_ = v;}
    unsigned int flags() const {return flags_;
    unsigned int flags() const {return flags_;
    unsigned int flags() const {return flags_;
    unsigned int flags() const {return flags_;

    void set_flag(unsigned int c) (flags_ |= c);
    void clear_flag(unsigned int c) (flags_ &= ~c);

    enum {
        INACTIVE = 1«0,
        INVISIBLE = 1«1,
        OUTPUT = 1«2,
        NOBORDER = 1«3,
        FORCEPOSITION = 1«4,
        NON_MODAL = 1«5,
        SHORTCUT_LABEL = 1«6,
        CHANGED = 1«7,
        OVERRIDE = 1«8,

    Generated by Doxygen
VISIBLE_FOCUS = 1x9,
COPYED_LABEL = 1x10,
CLIP_CHILDREN = 1x11,
MENU_WINDOW = 1x12,
TOOLTIP_WINDOW = 1x13,
MODAL = 1x14,
NO_OVERLAY = 1x15,
GROUP_RELATIVE = 1x16,
COPYED_TOOLTIP = 1x17,
FULLSCREEN = 1x18,
MAC_USE_ACENTS_MENU = 1x19,
NEEDS_KEYBOARD = 1x20,
IMAGE_BOUND = 1x21,
DEIMAGE_BOUND = 1x22,
AUTO_DELETE_USER_DATA = 1x23,
MAXIMIZED = 1x24,
// Note to devs: add new FLTK core flags above this line (up to 1x28).

USERFLAG3 = 1x29,
USERFLAG2 = 1x30,
USERFLAG1 = 1x31

}$

void draw_box() const;
void draw_box(Fl_Boxtype t, Fl_Color c) const;
void draw_box(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const;
void draw_backdrop() const;

void draw_focus() const {
  draw_focus(box(), x(), y(), w(), h(), color());
}

void draw_focus(Fl_Boxtype t, int X, int Y, int W, int H) const {
  draw_focus(t, X, Y, W, H, color());
}

// See documentation in Fl_Widget.cxx
void draw_focus(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const;

void draw_label() const;
void draw_label(int, int, int, int) const;

public:

virtual ~Fl_Widget();
virtual void draw() = 0;

virtual int handle(int event);

int is_label_copied() const {
  return ((flags_ & COPIED_LABEL) ? 1 : 0);}

void needs_keyboard(bool needs) {
  if (needs) set_flag(NEEDS_KEYBOARD);
  else clear_flag(NEEDS_KEYBOARD);
}

bool needs_keyboard() const {
  return (flags_ & NEEDS_KEYBOARD);
}

Fl_Group * parent() const {
  return parent_;
}

void parent(Fl_Group * p) {
  parent_ = p; // for hacks only, use Fl_Group::add()
}

uchar type() const {
  return type_;
}

void type(uchar t) {
  type_ = t;
}

int x() const {
  return x_;
}

int y() const {
  return y_;
}

int w() const {
  return w_;
}

int h() const {
  return h_;
}

virtual void resize(int x, int y, int w, int h);

int damage_resize(int, int, int, int);

void position(int X, int Y) {
  resize(X, Y, w_, h_);
}

void size(int W, int H) {
  resize(x_, y_, W, H);
}

Fl_Align align() const {
  return label_.align_;
}

Generated by Doxygen
void align(Fl_Align alignment) {label_.align_ = alignment;}

Fl_Boxtype box() const {return (Fl_Boxtype)box_;

void box(Fl_Boxtype new_box) {box_ = new_box;}

Fl_Color color() const {return color_;

void color(Fl_Color bg) {color_ = bg;}

Fl_Color selection_color() const {return color2_;

void selection_color(Fl_Color a) {color2_ = a;

void color(Fl_Color bg, Fl_Color sel) {color_ = bg; color2_ = sel;

const char* label() const {return label_.value;

void label(const char* text);

void copy_label(const char* new_label);

void label(Fl_Labeltype a, const char* b) {label_.type = a; label_.value = b;

Fl_Labeltype labeltype() const {return (Fl_Labeltype)label_.type;

void labeltype(Fl_Labeltype a) {label_.type = a;

Fl_Color labelcolor() const {return label_.color;

void labelcolor(Fl_Color c) {label_.color = c;

Fl_Font labelfont() const {return label_.font;

void labelfont(Fl_Font f) {label_.font = f;

Fl_Fontsize labelsize() const {return label_.size;

void labelsize(Fl_Fontsize pix) {label_.size = pix;

Fl_Image* image() {return label_.image;

const Fl_Image* image() const {return label_.image;

void image(Fl_Image* img);

void image(Fl_Image& img);

void bind_image(Fl_Image* img);

void bind_image(Fl_Image& img);

int image_bound() const {return ((flags_ & IMAGE_BOUND) ? 1 : 0);

void bind_deimage(Fl_Image* img);

int deimage_bound() const {return ((flags_ & DEIMAGE_BOUND) ? 1 : 0);

void bind_deimage(Fl_Image& img);

void bind_image(int f) {if (f) set_flag(IMAGE_BOUND); else clear_flag(IMAGE_BOUND);

void bind_deimage(int f) {if (f) set_flag(DEIMAGE_BOUND); else clear_flag(DEIMAGE_BOUND);

void bind_image(Fl_Image* img);

void bind_deimage(Fl_Image* img);

int tooltip_bound() const {return ((flags_ & TOOLTIP_BOUND) ? 1 : 0);

void bind_tooltip(int f) {if (f) set_flag(TOOLTIP_BOUND); else clear_flag(TOOLTIP_BOUND);

void bind_tooltip(Fl_Image* img);

void bind_tooltip(Fl_Image& img);

void bind_tooltip(int f) {if (f) set_flag(TOOLTIP_BOUND); else clear_flag(TOOLTIP_BOUND);

void bind_tooltip(Fl_Image* img);

void bind_tooltip(Fl_Image& img);

void callback(Fl_Callback* cb, void* p) {

void callback(Fl_Callback* cb, Fl_Callback_User_Data* p, bool auto_free) {

void callback(Fl_Callback* cb) {callback_ = cb;

Fl_Callback_p callback() const {return callback_;

void callback(Fl_Callback* cb, void* p) {

callback_ = cb;

user_data(p);

callback_ = cb;

user_data(p, auto_free);

callback_ = cb;

callback_ = cb;

Generated by Doxygen
void callback(Fl_Callback * cb) {
    callback_ = (Fl_Callback *)(fl_intptr_t)(cb);
}

void callback(Fl_Callback1 * cb, long p = 0) {
    callback_ = (Fl_Callback *)(fl_intptr_t)(cb);
    user_data((void *)(fl_intptr_t)p);
}

void* user_data() const {return user_data_;

void user_data(void * v);

void user_data(Fl_Callback_User_Data * v, bool auto_free);

long argument() const {return (long)(fl_intptr_t)user_data_;

void argument(long v) {user_data((void *)(fl_intptr_t)v);

Fl_When when() const {return (Fl_When)when_;

void when(uchar i) {when_ = i;

unsigned int visible() const {return !(flags_&INVISIBLE);

int visible_r() const;

virtual void show();

virtual void hide();

void set_visible() {flags_ &= ~INVISIBLE;}

void clear_visible() {flags_ |= INVISIBLE;}

unsigned int active() const {return !(flags_&INACTIVE);

int active_r() const;

void activate();

void deactivate();

unsigned int output() const {return (flags_&OUTPUT);

void set_output() {flags_ |= OUTPUT;}

void clear_output() {flags_ &= ~OUTPUT;}

unsigned int takesevents() const {return !(flags_&(INACTIVE|INVISIBLE|OUTPUT));

unsigned int changed() const {return flags_ & CHANGED;

void set_changed() {flags_ |= CHANGED;}

void clear_changed() {flags_ &= ~CHANGED;}

int take_focus();

void set_visible_focus() { flags_ |= VISIBLE_FOCUS; }

void clear_visible_focus() { flags_ &= ~VISIBLE_FOCUS; }

void visible_focus(int v) { if (v) set_visible_focus(); else clear_visible_focus(); }

unsigned int visible_focus() const { return flags_ & VISIBLE_FOCUS; }

*/

static void default_callback(Fl_Widget *widget, void *data);

void do_callback(Fl_Callback_Reason reason=FL_REASON_UNKNOWN) {do_callback(this, user_data_,

void do_callback(Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN) {
    do_callback(widget, (void*){fl_intptr_t}arg, reason);

} */

void do_callback(Fl_Widget *widget, void *arg = 0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN);

int testShortcut();

*/ Internal use only. */

static unsigned int labelShortcut(const char *t);
01068 /* Internal use only. */
01069 static int testShortcut(const char*, const bool require_alt = false);
01070 /* Internal use only. */
01071 void _set_fullscreen() {flags_ |= FULLSCREEN;}
01072 void _clear_fullscreen() {flags_ &= ~FULLSCREEN;}
01073 int contains(const Fl_Widget *w) const ;
01075 void redraw();
01076 void redraw_label();
01078 int contains(const Fl_Widget *wgt) const {return wgt ? wgt->contains(this) : 0;}
01079 void _set_fullscreen();
01080 void _clear_fullscreen();
01087 int inside(const Fl_Widget *wgt) const {return wgt ? wgt->contains(this) : 0;}
01088 uchar damage() const {return damage_;}
01089 uchar damage(const uchar c = 0) {damage_ = c;}
01109 uchar damage(uchar c);
01125 uchar damage(uchar c, int x, int y, int w, int h);
01133 void draw_label(int, int, int, int, Fl_Align) const;
01134 void measure_label(int& ww, int& hh) const {label_.measure(ww, hh);}
void traverse(Fl_Widget *widget); // finds subwindows of widget and prints them

protected:
int x_offset;
int y_offset;
Fl_Widget_Surface(Fl_Graphics_Driver *d);

public:
virtual void translate(int x, int y);
virtual void untranslate();
draw(Fl_Widget * widget, int delta_x = 0, int delta_y = 0);
draw_decorated_window(Fl_Window *win, int x_offset = 0, int y_offset = 0);
print_window_part(Fl_Window *win, int x, int y, int w, int h, int delta_x = 0, int delta_y = 0);
virtual int printable_rect(int *w, int *h);
origin(int x, int y);
origin(int *x, int *y);

#endif /* Fl_Widget_Surface_h */
#define FL_DOUBLE_WINDOW 0xF1

class Fl_X;
class Fl_Window_Driver;
class Fl_RGB_Image;
class Fl_Double_Window;

class FL_EXPORT Fl_Window : public Fl_Group {
  friend class Fl_X;
  friend class Fl_Window_Driver;
  private:
    static char *default_xclass_;
    static char show_next_window_iconic_; // 1 means create next window in iconic form

  int no_fullscreen_x;
  int no_fullscreen_y;
  int no_fullscreen_w;
  int no_fullscreen_h;
  int fullscreen_screen_top;
  int fullscreen_screen_bottom;
  int fullscreen_screen_left;
  int fullscreen_screen_right;

  // TODO: it would make sense to merge the use of Fl_X and Fl_Window_Driver, maybe simply by
  // TODO: deriving Fl_Window_Driver from Fl_X. However, there are a lot of historic kludges
  // TODO: for some platforms around Fl_X.
  Fl_X *flx_; // points at the system-specific stuff, but exists only after the window is mapped
  Fl_Window_Driver *pWindowDriver; // points at the system-specific stuff at window creation time

  const char * iconlabel_;
  char * xclass_;

  // private size_range stuff:
  int xmed_, ymed_, maxw_, maxh_;
  int dw_, dh_, aspect_;
  uchar size_range_set_; // true (1) if size_range() has been set or calculated

  // cursor stuff
  Fl_Cursor cursor_default;

  void _Fl_Window(); // constructor inners

  // unimplemented copy ctor and assignment operator
  Fl_Window(const Fl_Window&);
  Fl_Window& operator=(const Fl_Window&);

  void is_maximized_(bool b);

  protected:

  static Fl_Window *current_;

  void draw() FL_OVERRIDE;
  virtual void flush();

  void force_position(int force) {
    if (force) set_flag(FORCE_POSITION);
    else clear_flag(FORCE_POSITION);
  }
  int force_position() const { return ((flags() & FORCE_POSITION)?1:0); }

  void free_icons();

  void default_size_range(); // calculate size_range() if not set explicitly
  int is_resizable(); // calculate size_range() and return whether this is resizable

  public:

  Fl_Window(int w, int h, const char *title = 0);
  Fl_Window(int x, int y, int w, int h, const char *title = 0);
  virtual ~Fl_Window();

  int handle(int) FL_OVERRIDE;

  void resize(int X,int Y,int W,int H) FL_OVERRIDE;
  void border(int b);
  void clear_border() {set_flag(NOBORDER);}
  unsigned int border() const {return !(flags() & NOBORDER);}
  void set_override() {set_flag(NOBORDER|OVERRIDE);}
  unsigned int override() const { return flags()&OVERRIDE; }
  void set_modal() {set_flag(MODAL);}
  unsigned int modal() const {return flags() & MODAL;}
  void set_non_modal() {set_flag(NON_MODAL);}
  unsigned int non_modal() const {return flags() & (NON_MODAL|MODAL);}

  void clear_modal_states() {clear_flag(NON_MODAL | MODAL);}

  void set_menu_window() {set_flag(MENU_WINDOW);}

  void default_size_range(); // calculate size_range() if not set explicitly
  int is_resizable(); // calculate size_range() and return whether this is resizable
}
unsigned int menu_window() const { return flags() & MENU_WINDOW; }
void set_tooltip_window()
{ set_flag(TOOLTIP_WINDOW);
clear_flag(MENU_WINDOW); }
unsigned int tooltip_window() const { return flags() & TOOLTIP_WINDOW; }
void set_tooltip_window() { set_flag(TOOLTIP_WINDOW);
clear_flag(MENU_WINDOW); }
unsigned int tooltip_window() const { return flags() & TOOLTIP_WINDOW; }
void hotspot(int x, int y, int offscreen = 0);
void hotspot(const Fl_Widget*, int offscreen = 0);
void hotspot(const Fl_Widget & p, int offscreen = 0) { hotspot(&p, offscreen); }
void free_position() { clear_flag(FORCE_POSITION); }
void size_range(int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0);
const char * label() const { return Fl_Widget::label(); }
const char * iconlabel() const { return iconlabel_; }
void label(const char * label);
void iconlabel(const char *);
void label(const char * label, const char* iconlabel); // platform dependent
void copy_label(const char * a);
static void default_xclass(const char *);
static const char *default_xclass();
const char * xclass() const;
xclass(const char * c);
static void default_icon(const Fl_RGB_Image *);
static void default_icons(const Fl_RGB_Image *[], int);
void icon(const Fl_RGB_Image *);
void icons(const Fl_RGB_Image *[], int);

static void default_icons(HICON big_icon, HICON small_icon);
icons(HICON big_icon, HICON small_icon);

const void * icon() const;
void icon(const void * ic);
int shown() { return flx_ != 0; }
public:
void iconize();

int x_root() const ;
int y_root() const ;

static Fl_Window *current();
void make_current();

void cursor(Fl_Cursor);
void cursor(const Fl_RGB_Image *, int, int);
void default_cursor(Fl_Cursor);

/* for legacy compatibility */
const void* icon() const;
void icon(const void * i);
int shown() { return flx_ != 0; }
void show() FL_OVERRIDE;
void hide() FL_OVERRIDE;
void show(int argc, char **argv);

// Enables synchronous show(), docs in Fl_Window.cxx
void wait_for_expose();

void fullscreen();
void fullscreen_off();
void fullscreen_off(int X, int Y, int W, int H);
unsigned int fullscreen_active() const { return flags() & FULLSCREEN; }
void fullscreenscreens(int top, int bottom, int left, int right);

void maximize();
void un_maximize();
unsigned int maximize_active() const { return flags() & MAXIMIZED; }

int decorated_w() const;
int decorated_h() const;

// Note: Doxygen docs in Fl_Widget.h to avoid redundancy.
00583  Fl_Window* as_window() FL_OVERRIDE { return this; }
00584  Fl_Window const* as_window() const FL_OVERRIDE { return this; }
00585
00589  virtual class Fl_Overlay_Window *as_overlay_window() {return 0L; }
00590
00594  virtual class Fl_Double_Window *as_double_window() {return 0L;}
00595
00596  void shape(const Fl_Image* img);
00597  void shape(const Fl_Image& b);
00598  const Fl_Image *shape();
00599  void draw_backdrop();
00600  int screen_num();
00601  void draw_screen_num(int screen_num);
00602  static bool is_a_rescale();
00603  fl_uintptr_t os_id();
00604
00612  static void show_next_window_iconic(char stat) {
00613    show_next_window_iconic_ = stat ? 1 : 0;
00614  }
00615
00623  static char show_next_window_iconic() {
00624    return show_next_window_iconic_
00625  }
00626  
00627  
00628  

13.181  Fl_Wizard.H

00001  //
00002  // Fl_Wizard widget definitions.
00003  //
00004  // Copyright 1999-2010 by Easy Software Products.
00005  // Copyright 2011-2020 by Bill Spitzak and others.
00006  //
00007  // This library is free software. Distribution and use rights are outlined in
00008  // the file "COPYING" which should have been included with this file. If this
00009  // file is missing or damaged, see the license at:
00010  //
00011  // https://www.fltk.org/COPYING.php
00012  //
00013  // Please see the following page on how to report bugs and issues:
00014  //
00015  // https://www.fltk.org/bugs.php
00016  //
00017  // File
00018  Fl_Wizard widget . */
00019
00020  //
00021  // Include necessary header files...
00022  //
00023  //
00024  #ifdef _Fl_Wizard_H_
00025  #define _Fl_Wizard_H_
00026  #define _Fl_Wizard_H_
00027  #include <FL/Fl_Group.H>
00028  
00029  class FlEXPORT Fl_Wizard : public Fl_Group {
00030    Fl_Widget *value_;
00031  
00032  protected:
00033  
00034  void draw() FL_OVERRIDE;
00035  
00036  public:
00037  Fl_Wizard(int, int, int, int, const char * = 0);
00038    void next();
00039    void prev();
00040    Fl_Widget *value();
00041    void value(Fl_Widget *);
00042  
00043  
00044  #endif // _Fl_Wizard_H_

13.182  Fl_XBM_Image.H

00001  //
00002  // XBM image header file for the Fast Light Tool Kit (FLTK).
/* fl_xpm_image_class */

#ifndef Fl_XPM_Image_H
#define Fl_XPM_Image_H

#include "Fl_Pixmap.H"

class FL_EXPORT Fl_XPM_Image : public Fl_Pixmap {

public:

    Fl_XPM_Image(const char * filename);

};

#endif // !Fl_XPM_Image_H

13.184 forms.H

/* fl_forms_class */

#ifndef Fl_Forms_H
#define Fl_Forms_H

#include "fl_layout.H"

class FL_EXPORT Fl_Forms {

public:

    Fl_Forms();

};

#endif // !Fl_Forms_H

13.185 fl_layout.H

/* fl_layout_class */

#ifndef Fl_Layout_H
#define Fl_Layout_H

#include "Fl_Pixmap.H"

class FL_EXPORT Fl_Layout {

public:

    Fl_Layout();

};

#endif // !Fl_Layout_H

Generated by Doxygen
```c
#ifndef __FORMS_H__
#define __FORMS_H__

#include "Fl.H"
#include "Fl_Group.H"
#include "Fl_Window.H"
#include "fl_draw.H"

typedef Fl_Widget FL_OBJECT;
typedef Fl_Window FL_FORM;

// Random constants & symbols defined by forms.h file:
#ifndef NULL
#define NULL 0
#endif
#ifndef FALSE
#define FALSE 0
#define TRUE 1
#endif

#define FL_ON 1
#define FL_OK 1
#define FL_VALID 1
#define FL_PREEMPT 1
#define FL_AUTO 2
#define FL_WHEN_NEEDED_FL_AUTO
#define FL_OFF 0
#define FL_NONE 0
#define FL_CANCEL 0
#define FL_INVALID 0
#define FL_IGNORE -1
// #define FL_CLOSE -2 // this variable is never used in FLTK Forms. It is removed
// because it conflicts with the window FL_CLOSE event
#define FL_ALIGN_BESIDE FL_ALIGN_INSIDE

#define FL_PUP_TOGGLE 2 // FL_MENU_TOGGLE
#define FL_PUP_INACTIVE 1 // FL_MENU_INACTIVE
#define FL_NO_FRAME FL_NO_BOX
#define FL_ROUNDED3D_UPBOX FL_ROUND_UP_BOX
#define FL_ROUNDED3D_DOWNBOX FL_ROUND_DOWN_BOX
#define FL_OVAL3D_UPBOX FL_ROUND_UP_BOX
#define FL_OVAL3D_DOWNBOX FL_ROUND_DOWN_BOX

#define FL_MBUTTON1 1
#define FL_LEFTMOUSE 1
#define FL_MBUTTON2 2
#define FL_MIDDLEMOUSE 2
#define FL_MBUTTON3 3
#define FL_RIGHTMOUSE 3
```
00104 #define FL_MBUTTON4 4
00105 #define FL_MBUTTON5 5
00106
00107 #define FL_INVALID_STYLE 255
00108 #define FL_NORMAL_STYLE FL_HELVETICA
00109 #define FL_BOLD_STYLE FL_HELVETICA_BOLD
00110 #define FL_ITALIC_STYLE FL_HELVETICA_ITALIC
00111 #define FL_BOLDITALIC_STYLE FL_HELVETICA_BOLD_ITALIC
00112 #define FL_FIXED_STYLE FL_COURIER
00113 #define FL_FIXEDBOLD_STYLE FL_COURIER_BOLD
00114 #define FL_FIXEDITALIC_STYLE FL_COURIER_ITALIC
00115 #define FL_FIXEDBOLDITALIC_STYLE FL_COURIER_BOLD_ITALIC
00116 #define FL_TIMES_STYLE FL_TIMES
00117 #define FL_TIMESBOLD_STYLE FL_TIMES_BOLD
00118 #define FL_TIMESITALIC_STYLE FL_TIMES_ITALIC
00119 #define FL_TIMESBOLDITALIC_STYLE FL_TIMES_BOLD_ITALIC
00120
00121 // hacks to change the labeltype() when passed to fl_set_object_lstyle():
00122 #define FL_SHADOW_STYLE (FL_SHADOW_LABEL«8)
00123 #define FL_ENGRAVED_STYLE (FL_ENGRAVED_LABEL«8)
00124 #define FL_EMBOSSED_STYLE (FL_EMBOSSED_LABEL«8)
00125
00126 // size values are different from XForms, match older Forms:
00127 #define FL_TINY_SIZE 8
00128 #define FL_SMALL_SIZE 11 // 10
00129 //define FL_NORMAL_SIZE 14 // 12
00130 #define FL_MEDIUM_SIZE 18 // 14
00131 #define FL_LARGE_SIZE 24 // 18
00132 #define FL_HUGE_SIZE 32 // 24
00133 #define FL_DEFAULT_SIZE 32 // 24
00134 #define FL_TINY_FONT FL_TINY_SIZE
00135 #define FL_SMALL_FONT FL_SMALL_SIZE
00136 #define FL_NORMAL_FONT FL_NORMAL_SIZE
00137 #define FL_MEDIUM_FONT FL_MEDIUM_SIZE
00138 #define FL_LARGE_FONT FL_LARGE_SIZE
00139 #define FL_HUGE_FONT FL_HUGE_SIZE
00140 #define FL_DEFAULT_FONT FL_DEFAULT_SIZE
00141 #define FL_TINY_FONT1 FL_TINY_FONT
00142 #define FL_SMALL_FONT1 FL_SMALL_FONT
00143
00144 #define FL_RETURN_END_CHANGED FL_WHEN_RELEASE
00145 #define FL_RETURN_CHANGED FL_WHEN_CHANGED
00146 #define FL_RETURN_END FL_WHEN_RELEASE_ALWAYS
00147 #define FL_RETURN_ALWAYS (FL_WHEN_CHANGED|FL_WHEN_NOT_CHANGED)
00148
00149 #define FL_BOUND_WIDTH 3
00150
00151 typedef int FL_Coord;
00152 typedef int FL_COLOR;
00153
00154 // fltk interaction:
00155
00156 #define FL_CMD_OPT void
00157 extern FL_EXPORT void fl_initialize(int*, char*[]), const char* = FL_CMD_OPT*, int);
00158 inline void fl_finish() {}
00159
00160 typedef void (*FL_IO_CALLBACK) (FL_SOCKET, void*);
00161 inline void fl_add_io_callback(int fd, short w, FL_IO_CALLBACK cb, void* v) {
00162 Fl::add_fd(fd, w, cb, v);
00163 Fl::add_io_callback(int fd, short w, FL_IO_CALLBACK cb, void* v) {
00164 Fl::remove_io_callback(int fd, short, FL_IO_CALLBACK) {
00165 Fl::remove_fd(fd);} // removes all the callbacks!
00166
00167 // type of callback is different and no "id" number is returned:
00168 inline void fl_add_timeout(long msec, void (*cb)(void*), void* v) {
00169 Fl::add_timeout(msec*.001, cb, v);
00170 Fl::add_timeout(long msec, void (*cb)(void*), void* v) {
00171 inline void fl_remove_timeout(int) {}
00172
00173 // type of callback is different:
00174 inline void fl_set_idle_callback(void (*cb)()) [Fl::set_idle(cb)];
00175 FL_EXPORT Fl_Widget* fl_do_forms(void);
00176 FL_EXPORT Fl_Widget* fl_check_forms();
00177 inline Fl_Widget* fl_do_forms(void) {return fl_do_forms();}
00178 inline Fl_Widget* fl_check_forms(void) {return fl_check_forms();}
00179
00180 // because of new redraw behavior, these are no-ops:
00181 inline void fl_freeze_object(Fl_Widget*) {} [Fl::freeze_object(Fl_Widget*)];
00182 inline void fl_unfreeze_object(Fl_Widget*) {} [Fl::unfreeze_object(Fl_Widget*)];
00183 inline void fl_freeze_form(FL_Widget*) {} [Fl::freeze_form(Fl_Widget*)];
00184 inline void fl_unfreeze_form(FL_Widget*) {} [Fl::unfreeze_form(Fl_Widget*)];
00185 inline void fl_freeze_all_forms() {} [Fl::freeze_all_forms();]
00186 inline void fl_unfreeze_all_forms() {} [Fl::unfreeze_all_forms();]
00187
00188 inline void fl_set_focus_object(Fl_Widget*, Fl_Widget* o) [Fl::set_focus(Fl_Widget* o)];
00189 inline void fl_reset_focus_object(Fl_Widget* o) [Fl::reset_focus(Fl_Widget* o)];
00190 #define fl_set_object_focus fl_set_focus_object
00191
00192 Generated by Doxygen
// void fl_set_form_atclose(Fl_Window*w, int (*cb)(Fl_Window*,void*,void*));
// void fl_set_atclose(int (*cb)(Fl_Window*,void*),void*);  
// fl_set_form_atactivate/atdeactivate not implemented!

// Fl_Widget:
inline void fl_set_object_boxtype(Fl_Widget* o, Fl_Boxtype a) {o->box(a);}
inline void fl_set_object_lsize(Fl_Widget* o,int s) {o->labelsize(s);}
inline void fl_set_object_lstyle(Fl_Widget* o, int a) {o->labelfont((Fl_Font)(a&0xff)); o->labeltype((Fl_Labeltype)(a»8));}
inline void fl_set_object_lcol(Fl_Widget* o, Fl_Color a) {o->labelcolor(a);}
define fl_set_object_lcolor fl_set_object_lcol
inline void fl_set_object_lalign(Fl_Widget* o, Fl_Align a) {o->align(a);}
define fl_set_object_align fl_set_object_lalign
inline void fl_set_object_color(Fl_Widget* o, Fl_Color a,Fl_Color b) {o->color(a,b);}
inline void fl_set_object_label(Fl_Widget* o, const char* a) {o->label(a); o->redraw();}
inline void fl_set_object_position(Fl_Widget* o, int x, int y) {o->position(x,y);}
// things that appered in the demos a lot that I don’t emulate, but
// I did not want to edit out of all the demos...
inline int fl_get_border_width() {return 3;}
inline void fl_set_border_width(int) {}  
inline void fl_set_object_dblbuffer(Fl_Widget*, int) {}  
inline void fl_set_form_dblbuffer(Fl_Window*, int) {}  

// Fl_Window:
inline void fl_free_form(Fl_Window* x) {delete x;}
inline void fl_redraw_form(Fl_Window* f) {f->redraw();}
inline Fl_Window* fl_bgn_form(Fl_Boxtype b,int w,int h) {
  Fl_Window * g = new Fl_Window(w,h,0);
  g->box(b);
  return g;}
inline void fl_addto_form(Fl_Window* f) {f->begin();}
inline Fl_Group* fl_bgn_group() {return new Fl_Group(0,0,0,0,0);}
inline void fl_addto_group(Fl_Widget* o) {((Fl_Group* )o)->begin();}
define resizebox _ddfdesign_kludge()   
inline void fl_scale_form(Fl_Window* f, double x, double y) {
  f->resizable(f);  f->size(int(f->w()»x),int(f->h()»y));}
inline void fl_set_form_position(Fl_Window* f, int x, int y) {f->position(x,y);}
inline void fl_set_form_size(Fl_Window* f, int w, int h) {f->size(w,h);}
inline void fl_set_form_geometry(Fl_Window* f, int x, int y, int w, int h) {
  f->resize(x,y,w,h);}
define fl_set_initial_placement fl_set_form_geometry
inline void fl_adjust_form_size(Fl_Window*) {}

FL_EXPORT void fl_show_form(Fl_Window* f,int p,int b,const char* n);
#define resizebox _ddfdesign_kludge()   
inline void fl_add_object(Fl_Window* f, Fl_Widget* x) {f->add(x);}
inline void fl_insert_object(Fl_Widget* o, Fl_Widget* b) {b->parent()->insert(o,b);}
inline Fl_Window* FL_ObjWin(Fl_Widget* o) {return o->window();}
inline void fl_get_object_position(Fl_Widget* o, int x, int y) {
  x = o->x();  y = o->y();}
inline int fl_get_border_width() {return 3;}
inline int fl_get_object_position(Fl_Widget* o, int x, int y) {
  x = o->x();  y = o->y();}
#define resizebox _ddfdesign_kludge()   
inline void fl_scale_form(Fl_Window* f, double x, double y) {
  f->resizable(f);  f->size(int(f->w()»x),int(f->h()»y));}
inline void fl_set_form_position(Fl_Window* f, int x, int y) {f->position(x,y);}
inline void fl_set_form_size(Fl_Window* f, int w, int h) {f->size(w,h);}
inline void fl_set_form_geometry(Fl_Window* f, int x, int y, int w, int h) {
  f->resize(x,y,w,h);}
#define fl_set_initial_placement fl_set_form_geometry
inline void fl_adjust_form_size(Fl_Window*) {}
FL_PLACE_POSITION = 4, // fixed position, resizable
FL_PLACE_SIZE = 8, // fixed size, normal fltk behavior
FL_PLACE_GEOMETRY = 16, // fixed size and position
FL_PLACE_ASPECT = 32, // keep aspect ratio (ignored)
FL_PLACE_FULLSCREEN = 64, // fill screen
FL_PLACE_HOTSPOT = 128, // enables hotspot
FL_PLACE_ICONIC = 256, // iconic (ignored)
FL_FREE_SIZE = (1«14), // force resizable
FL_FIX_SIZE = (1«15) // force off resizable

#define FL_PLACE_FREE_CENTER (FL_PLACE_CENTER|FL_FREE_SIZE)
#define FL_PLACE_CENTERFREE (FL_PLACE_CENTER|FL_FREE_SIZE)
enum { // "b" argument values:
    FL_NOBORDER = 0,
    FL_FULLBORDER,
#endif
inline void fl_set_form_hotspot(Fl_Window* w, int x, int y) {w->hotspot(x,y);}
inline void fl_set_form_hotobject(Fl_Window* w, Fl_Widget* o) {w->hotspot(o);}
extern FL_EXPORT char fl_flip; // in forms.C
inline void fl_flip_yorigin() {fl_flip = 1;}
#define fl_prepare_form_window fl_show_form
inline void fl_show_form_window(Fl_Window*) {}
inline void fl_raise_form(Fl_Window* f) {f->show();}
inline void fl_hide_form(Fl_Window* f) {f->hide();}
extern FL_EXPORT char fl_modal_next; // in forms.C
inline void fl_activate_all_forms() {}
inline void fl_deactivate_all_forms() {fl_modal_next = 1;}
inline void fl_deactivate_form(Fl_Window*w) {w->deactivate();
inline void fl_activate_form(Fl_Window*w) {w->activate();
inline void fl_set_form_title(Fl_Window* f, const char* s) {f->label(s);}
inline void fl_title_form(Fl_Window* f, const char* s) {f->label(s);}
typedef void (*Forms_FormCB)(Fl_Widget*);
inline void fl_set_form_callback(Fl_Window* f, Forms_FormCB c) {f->callback(c);}
#define fl_set_form_call_back fl_set_form_callback
inline void fl_init() {}
FL_EXPORT void fl_set_graphics_mode(int,int);
inline int fl_form_is_visible(Fl_Window* f) {return f->visible();}
inline int fl_mouse_button() {return Fl::event_button();}
#define fl_free free
#define fl_malloc malloc
#define fl_calloc calloc
#define fl_realloc realloc

// Drawing functions. Only usable inside an Fl_Free object?
inline void fl_drw_box(Fl_Boxtype b, int x, int y, int w, int h, Fl_Color bgc, int=3) {
    fl_draw_box(b,x,y,w,h,bgc);
}
inline void fl_drw_frame(Fl_Boxtype b, int x, int y, int w, int h, Fl_Color bgc, int=3) {
    fl_draw_box(b,x,y,w,h,bgc);
}
inline void fl_drw_text(Fl_Align align, int x, int y, int w, int h, Fl_Color fgcolor, int size, Fl_Font style,
    const char* s) {
    fl_font(style,size);
    fl_color(fgcolor);
    fl_draw(s,x,y,w,h,align);
}
inline void fl_drw_text_beside(Fl_Align align, int x, int y, int w, int h, Fl_Color fgcolor, int size, Fl_Font style,
    const char* s) {
    fl_font(style,size);
    fl_color(fgcolor);
    fl_draw(s,x,y,w,h,align);
    fl_draw(s,x,y,w,h,align);
}
// this does not work except for CENTER...
inline void fl_drw_text_beside(Fl_Align align, int x, int y, int w, int h, Fl_Color fgcolor, int size, Fl_Font style,
    const char* s) {
    fl_font(style,size);
    fl_color(fgcolor);
    fl_draw(s,x,y,w,h,align);
    fl_draw(s,x,y,w,h,align);
}
// Drawing functions. Only usable inside an Fl_Free object?
inline void fl_drw_box(Fl_Boxtype b, int x, int y, int w, int h, Fl_Color bgc, int=3) {
    fl_draw_box(b,x,y,w,h,bgc);
}
inline void fl_drw_frame(Fl_Boxtype b, int x, int y, int w, int h, Fl_Color bgc, int=3) {
    fl_draw_box(b,x,y,w,h,bgc);
}
inline void fl_drw_text(Fl_Align align, int x, int y, int w, int h, Fl_Color fgcolor, int size, Fl_Font style,
    const char* s) {
    fl_font(style,size);
    fl_color(fgcolor);
    fl_draw(s,x,y,w,h,align);
    fl_draw(s,x,y,w,h,align);
}
// this does not work except for CENTER...
inline void fl_drw_text_beside(Fl_Align align, int x, int y, int w, int h, Fl_Color fgcolor, int size, Fl_Font style,
    const char* s) {
    fl_font(style,size);
    fl_color(fgcolor);
    fl_draw(s,x,y,w,h,align);
    fl_draw(s,x,y,w,h,align);
}
inline void fl_drw_box(Fl_Boxtype b, int x, int y, int w, int h, Fl_Color bgc, int=3) {
    fl_draw_box(b,x,y,w,h,bgc);
}
inline void fl_drw_frame(Fl_Boxtype b, int x, int y, int w, int h, Fl_Color bgc, int=3) {
    fl_draw_box(b,x,y,w,h,bgc);
}
inline void fl_drw_text(Fl_Align align, int x, int y, int w, int h, Fl_Color fgcolor, int size, Fl_Font style,
    const char* s) {
    fl_font(style,size);
    fl_color(fgcolor);
    fl_draw(s,x,y,w,h,align);
    fl_draw(s,x,y,w,h,align);
}
// this does not work except for CENTER...

inline Fl_Widget* fl_add_new(Fl_Widget* p) {return p;}
inline Fl_Widget* fl_add_new(uchar t, Fl_Widget* p) {p->type(t); return p;}

#define forms_constructor(type, name)
inline type* name(uchar t, int x, int y, int w, int h, const char* l) {
    return (type*)(fl_add_new(t, new type(x, y, w, h, l)));
}
#define forms_constructort(type, name)
inline type* name(uchar t, int x, int y, int w, int h, const char* l) {
    return (type*)(fl_add_new(new type(t, x, y, w, h, l)));
}
#define forms_constructorb(type, name)
inline type* name(Fl_Boxtype t, int x, int y, int w, int h, const char* l) {
    return (type*)(fl_add_new(new type(t, x, y, w, h, l)));
}

#include "Fl_FormsBitmap.H"
#define FL_NORMAL_BITMAP FL_NO_BOX
forms_constructorb(Fl_FormsBitmap, fl_add_bitmap)
inline void fl_set_bitmap_data(Fl_Widget* o, int w, int h, const uchar* b) {
    ((Fl_FormsBitmap*)o)->set(w, h, b);
}

#include "Fl_FormsPixmap.H"
#define FL_NORMAL_PIXMAP FL_NO_BOX
forms_constructorb(Fl_FormsPixmap, fl_add_pixmap)
inline void fl_set_pixmap_data(Fl_Widget* o, char*const* b) {
    ((Fl_FormsPixmap*)o)->set(b);
}
inline void fl_set_pixmap_file(Fl_Widget*, const char*);
inline void fl_set_pixmap_align(Fl_Widget* o, Fl_Align a, int, int);
#define fl_append_browser fl_addto_browser_chars
inline void fl_insert_browser_line(Fl_Widget* o, int n, const char* s) {
    ((Fl_Browser*)o)->insert(n,s);
}
inline void fl_delete_browser_line(Fl_Widget* o, int n) {
    ((Fl_Browser*)o)->remove(n);
}
inline void fl_replace_browser_line(Fl_Widget* o, int n, const char* s) {
    ((Fl_Browser*)o)->replace(n,s);
}
inline char* fl_get_browser_line(Fl_Widget* o, int n) {
    return (char*)((Fl_Browser*)o)->text(n);
}
inline int fl_load_browser(Fl_Widget* o, const char* f) {
    return ((Fl_Browser*)o)->load(f);
}
inline void fl_set_browser_topline(Fl_Widget* o, int n) {
    ((Fl_Browser*)o)->topline(n);
}
inline int fl_get_browser_topline(Fl_Widget* o) {
    return ((Fl_Browser*)o)->topline();
}
inline void fl_set_browser_fontsize(Fl_Widget* o, int s) {
    ((Fl_Browser*)o)->textsize(s);
}
inline void fl_set_browser_fontstyle(Fl_Widget* o, Fl_Font f) {
    ((Fl_Browser*)o)->textfont(f);
}
inline void fl_set_browser_specialkey(Fl_Widget* o, char c) {
    ((Fl_Browser*)o)->format_char(c);
}
inline void fl_set_browser_vscrollbar(Fl_Widget*, int);
// inline void fl_set_browser_xoffset(Fl_Widget*, FL_Coord);
inline void fl_set_browser_scrollbarsize(Fl_Widget*, int, int);
inline void fl_setdisplayed_browser_line(Fl_Widget* o, int n, int i) {
  ((Fl_Browser*)o)->display(n,i);
}
inline int fl_isdisplayed_browser_line(Fl_Widget* o, int n) {
  return ((Fl_Browser*)o)->displayed(n);
}
#include "FL_Button.H"

#define FL_NORMAL_BUTTON 0
#define FL_TOUCH_BUTTON 4
#define FL_INPUT_BUTTON 5
#define FL_RETURN_BUTTON 6
#define FL_HIDDEN_RET_BUTTON 7
#define FL_PUSH_BUTTON FL_TOGGLE_BUTTON
#define FL_MENU_BUTTON 9

FL_EXPORT Fl_Button* fl_add_button(uchar t,int x,int y,int w,int h,const char* l);
inline int fl_get_button(Fl_Widget* b) {return ((Fl_Button*)b)->value();}
inline void fl_set_button(Fl_Widget* b, int v) {((Fl_Button*)b)->value(v);}
inline int fl_get_button_numb(Fl_Widget*) {return Fl::event_button();}
inline void fl_set_button_shortcut(Fl_Widget* b, const char* s,int=0) {
  ((Fl_Button *)b)->shortcut(s);
} // define fl_set_object_shortcut(b,s) fl_set_button_shortcut(b,s)
#include "Fl_Light_Button.H"
forms_constructor(Fl_Light_Button, fl_add_lightbutton)
#include "Fl_Round_Button.H"
forms_constructor(Fl_Round_Button, fl_add_roundbutton)
forms_constructor(Fl_Round_Button, fl_add_round3dbutton)
#include "Fl_Check_Button.H"
forms_constructor(Fl_Check_Button, fl_add_checkbutton)
inline Fl_Widget* fl_add_bitmapbutton(int t,int x,int y,int w,int h,const char* l) {Fl_Widget* o = fl_add_button(t,x,y,w,h,l); return o;}
inline void fl_set_bitmapbutton_data(Fl_Widget* o,int a,int b,uchar* c) {
  (new Fl_Bitmap(c,a,b))->label(o); // does not delete old Fl_Bitmap!
}
inline Fl_Widget* fl_add_pixmapbutton(int t,int x,int y,int w,int h,const char* l) {Fl_Widget* o = fl_add_button(t,x,y,w,h,l); return o;}
inline void fl_set_pixmapbutton_data(Fl_Widget* o, const char*const* c) {
  (new Fl_Pixmap(c))->label(o); // does not delete old Fl_Pixmap!
}

// Fl_Canvas object not yet implemented!
#include "Fl_Chart.H"
forms_constructor(Fl_Chart, fl_add_chart)
inline void fl_clear_chart(Fl_Widget* o) {
  ((Fl_Chart*)o)->clear();
}
inline void fl_add_chart_value(Fl_Widget* o, double v, const char* s, uchar c) {
  ((Fl_Chart*)o)->add(v,s,c);
}
inline void fl_insert_chart_value(Fl_Widget* o, int i, double v, const char* s, uchar c) {
  ((Fl_Chart*)o)->insert(i,v,s,c);
}
inline void fl_replace_chart_value(Fl_Widget* o, int i, double v, const char* s, uchar c) {
  ((Fl_Chart*)o)->replace(i,v,s,c);
}
inline void fl_set_chart_bounds(Fl_Widget* o, double a, double b) {
  ((Fl_Chart*)o)->bounds(a,b);
}
inline void fl_set_chart_maxnumb(Fl_Widget* o, int v) {
  ((Fl_Chart*)o)->maxsize(v);
}
inline void fl_set_chart_autosize(Fl_Widget* o, int v) {
  ((Fl_Chart*)o)->autosize(v);
}
inline void fl_set_chart_style(Fl_Widget* o, Fl_Font v) {
  ((Fl_Chart*)o)->textfont(v);
}
inline void fl_set_chart_lsize(Fl_Widget* o, Fl_Font v) {
  ((Fl_Chart*)o)->textsize(v);
}
inline void fl_set_chart_lcolor(Fl_Widget* o, Fl_Color v) {
  ((Fl_Chart*)o)->textcolor(v);
}
#define fl_set_chart_lcol fl_set_chart_lcolor
#include "Fl_Choice.H"
forms_constructor(Fl_Choice, fl_add_choice)
inline void fl_clear_choice(Fl_Widget* o) {
  ((Fl_Choice *)o)->clear();
}
inline void fl_addto_choice(Fl_Widget* o, const char* s) {
  ((Fl_Choice *)o)->add(s);
}
inline void fl_replace_choice(Fl_Widget* o, int i, const char* s) {
  ((Fl_Choice*)o)->replace(i-1,s);
}
inline void fl_delete_choice(Fl_Widget* o, int i) {
  ((Fl_Choice*)o)->remove(i-1);
}
inline void fl_set_choice(Fl_Widget* o, int i) {
    ((Fl_Choice*)o)->value(i-1);
}

// inline void fl_set_choice_text(Fl_Widget*, const char*);

inline int fl_get_choice(Fl_Widget* o) {
    return ((Fl_Choice*)o)->value()+1;
}

// inline const char* fl_get_choice_item_text(Fl_Widget*, int);

inline const char* fl_get_choice_text(Fl_Widget* o) {
    return ((Fl_Choice *)o)->text();
}

inline void fl_set_choice_fontsize(Fl_Widget* o, int x) {
    ((Fl_Choice *)o)->textsize(x);
}

inline void fl_set_choice_fontstyle(Fl_Widget* o, Fl_Font x) {
    ((Fl_Choice *)o)->textfont(x);
}

// inline void fl_set_choice_item_mode(Fl_Widget*, int, unsigned);

// inline void fl_set_choice_item_shortcut(Fl_Widget*, int, const char*);

#include "Fl_Clock.H"

forms_constructor(Fl_Clock, fl_add_clock)

inline void fl_get_clock(Fl_Widget* o, int* h, int* m, int* s) {
    *h = ((Fl_Clock*)o)->hour();
    *m = ((Fl_Clock*)o)->minute();
    *s = ((Fl_Clock*)o)->second();
}

#include "Fl_Counter.H"

forms_constructor(Fl_Counter, fl_add_counter)

inline void fl_set_counter_value(Fl_Widget* o, double v) {
    ((Fl_Counter *)o)->value(v);
}

inline void fl_set_counter_bounds(Fl_Widget* o, double a, double b) {
    ((Fl_Counter*)o)->bounds(a, b);
}

inline void fl_set_counter_step(Fl_Widget* o, double a, double b) {
    ((Fl_Counter*)o)->step(a, b);
}

inline void fl_set_counter_precision(Fl_Widget* o, int v) {
    ((Fl_Counter*)o)->precision(v);
}

inline void fl_set_counter_return(Fl_Widget* o, int i) {
    ((Fl_Counter*)o)->when((Fl_When)(i|FL_WHEN_RELEASE));
}

inline void fl_set_counter_angles(Fl_Widget* o, int a, int b) {
    ((Fl_Counter*)o)->angles(a, b);
    // ((Fl_Counter*)o)->direction(d);
}

inline void fl_set_dial_value(Fl_Widget* o, double v) {
    ((Fl_Dial *)o)->value(v);
}

inline double fl_get_dial_value(Fl_Widget* o) {
    return ((Fl_Dial *)o)->value();
}

inline void fl_set_dial_bounds(Fl_Widget* o, double a, double b) {
    ((Fl_Dial*)o)->bounds(a, b);
}

inline void fl_get_dial_bounds(Fl_Widget* o, float* a, float* b) {
    *a = float(((Fl_Dial*)o)->minimum());
    *b = float(((Fl_Dial*)o)->maximum());
}

//inline void fl_set_dial_cross(Fl_Widget* o, int);

// inline void fl_set_dial_direction(Fl_Widget* o, uchar d) {
//    ((Fl_Dial *)o)->direction(d);
//}

inline void fl_set_dial_step(Fl_Widget* o, double v) {
    ((Fl_Dial *)o)->step(v);
}

// Frames:

inline Fl_Widget* fl_add_frame(Fl_Boxtype i, int x, int y, int w, int h, const char * l) {
    return fl_add_box(i, x-3, y-3, w+6, h+6, l);
}

return fl_add_box(i, x-3, y-3, w+6, h+6, l);

// labelframe nyl

inline Fl_Widget* fl_add_labelframe(Fl_Boxtype i, int x, int y, int w, int h, const char * l) {
    Fl_Widget* o = fl_add_box(i, x-3, y-3, w+6, h+6, l);
    o->align(FL_ALIGN_LEFT);
13.184 forms.H

```
#include "Fl_Free.H"
inline Fl_Free*
fl_add_free(int t, double x, double y, double w, double h, const char* l, FL_HANDLEPTR hdl) {
    return (Fl_Free*)(fl_add_new(
        new Fl_Free(t, int(x), int(y), int(w), int(h), l, hdl)));
}

#include "fl_ask.H"
#include "fl_show_colormap.H"

inline int fl_show_question(const char* c, int = 0) {return fl_choice("%s", fl_no, fl_yes, 0L, c);}

FL_EXPORT void fl_show_message(const char *, const char *, const char *);
FL_EXPORT void fl_show_alert(const char *, const char *, const char *, int=0);
FL_EXPORT void fl_show_input(const char *, const char *, int x=0) {return fl_input("%s", d, l);}
FL_EXPORT /*const*/ char *fl_show_simple_input(const char *label, const char *, const char *, int=0);
inline void fl_set_goodies_font(Fl_Font a, Fl_Fontsize b) {fl_message_font(a, b);}
#define fl_show_messages fl_message
inline int fl_show_choices(const char *c, int n, const char *b1, const char *b2, const char * b3, int) {
    return fl_show_choice(0, c, 0, n, b1, b2, b3);
}

#include "filename.H"
#include "Fl_File_Chooser.H"
inline int do_matching(char*, const char*) {return fl_filename_match(a, b);}

// Forms-compatible file chooser (implementation in fselect.C):
FL_EXPORT char* fl_show_file_selector(const char* message, const char* dir, const char*, int);
FL_EXPORT char* fl_get_directory();
FL_EXPORT char* fl_get_pattern();
FL_EXPORT char* fl_get_filename();

#include "Fl_Input.H"
forms_constructor(Fl_Input, fl_add_input)
inline void fl_set_input(Fl_Widget*, const char*) {
    ((Fl_Input*)o)->value(v);
}
inline void fl_set_input_return(Fl_Widget*, int x) {
    ((Fl_Input*)o)->when((Fl_When)(x | FL_WHEN_RELEASE));
}
inline void fl_set_input_color(Fl_Widget*, Fl_Color a, Fl_Color b) {
    ((Fl_Input*)o)->textcolor(a);
    ((Fl_Input*)o)->cursor_color(b);
}
inline void fl_set_input_cursorpos(Fl_Widget*, int x, int y) {
    ((Fl_Input*)o)->insert_position(x);
}
inline int fl_get_input_cursorpos(Fl_Widget*, int x, int y) {
    return ((Fl_Input*)o)->insert_position(x);
}
inline int fl_get_input_table(Fl_Widget*, int); // types are not implemented, they all act like FL_PUSH_MENU:
#define DEF_PUSH_MENU 0
#define DEF_PUSHDOWN_MENU 1
#define DEF_TOUCH_MENU 2
forms_constructor(Fl_Menu_Button, fl_add_menu)
```

Generated by Doxygen
inline void fl_replace_menu_item(Fl_Widget* o, int i, const char* s) {
    ((Fl_Menu_Button *)o)->replace(i-1,s);
}
inline void fl_delete_menu_item(Fl_Widget* o, int i) {
    ((Fl_Menu_Button *)o)->remove(i-1);
}
inline void fl_set_menu_item_shortcut(Fl_Widget* o, int i, const char* s) {
    ((Fl_Menu_Button *)o)->shortcut(i-1,fl_old_shortcut(s));
}
inline void fl_set_menu_item_mode(Fl_Widget* o, int i, long x) {
    ((Fl_Menu_Button *)o)->mode(i-1,(int)x);
}
inline void fl_show_menu_symbol(Fl_Widget*, int i) {
    /* ((Fl_Menu_Button*)o)->show_menu_symbol(i); */
}
inline int fl_get_menu(Fl_Widget* o) {
    return ((Fl_Menu_Button *)o)->value()+1;
}
inline const char* fl_get_menu_item_text(Fl_Widget* o, int i) {
    return ((Fl_Menu_Button *)o)->text(i);
}
inline int fl_get_menu_maxitems(Fl_Widget* o) {
    return ((Fl_Menu_Button*)o)->size();
}
inline int fl_get_menu_item_mode(Fl_Widget* o, int i) {
    return ((Fl_Menu_Button*)o)->mode(i);
}
inline const char* fl_get_menu_text(Fl_Widget* o) {
    return ((Fl_Menu_Button *)o)->text();
}

#include "Fl_Positioner.H"
define FL_NORMAL_POSITIONER 0
forms_constructor(Fl_Positioner, fl_add_positioner)
inline void fl_set_positioner_xvalue(Fl_Widget* o, double v) {
    ((Fl_Positioner *)o)->xvalue(v);
}
inline double fl_get_positioner_xvalue(Fl_Widget* o) {
    return ((Fl_Positioner*)o)->xvalue();
}
inline void fl_set_positioner_xbounds(Fl_Widget* o, double a, double b) {
    ((Fl_Positioner*)o)->xbounds(a,b);
}
inline void fl_get_positioner_xbounds(Fl_Widget* o, float* a, float* b) {
    *a = float(((Fl_Positioner*)o)->xminimum());
    *b = float(((Fl_Positioner*)o)->xmaximum());
}
inline void fl_set_positioner_xstep(Fl_Widget* o, double v) {
    ((Fl_Positioner*)o)->xstep(v);
}
inline void fl_set_positioner_yvalue(Fl_Widget* o, double v) {
    ((Fl_Positioner *)o)->yvalue(v);
}
inline double fl_get_positioner_yvalue(Fl_Widget* o) {
    return ((Fl_Positioner*)o)->yvalue();
}
inline void fl_set_positioner_ybounds(Fl_Widget* o, double a, double b) {
    ((Fl_Positioner*)o)->ybounds(a,b);
}
inline void fl_get_positioner_ybounds(Fl_Widget* o, float* a, float* b) {
    *a = float(((Fl_Positioner*)o)->yminimum());
    *b = float(((Fl_Positioner*)o)->ymaximum());
}
inline void fl_set_positioner_ystep(Fl_Widget* o, double v) {
    ((Fl_Positioner*)o)->ystep(v);
}
inline void fl_set_positioner_yvalue(Fl_Widget* o, int v) {
    ((Fl_Positioner*)o)->yvalue(v);
}

#include "Fl_Slider.H"
define FL_HOR_BROWSER_SLIDER FL_HOR_SLIDER
#define FL_VERT_BROWSER_SLIDER FL_VERT_SLIDER
forms_constructor(Fl_Slider, fl_add_slider)
inline void fl_set_slider_value(Fl_Widget* o, double v) {
    ((Fl_Slider *)o)->value(v);
}
inline double fl_get_slider_value(Fl_Widget* o) {
    return ((Fl_Slider*)o)->value();
}
inline void fl_set_slider_bounds(Fl_Widget* o, double a, double b) {
    ((Fl_Slider*)o)->bounds(a,b);
}
inline void fl_get_slider_bounds(Fl_Widget* o, float* a, float* b) {
    *a = float(((Fl_Slider*)o)->minimum());
    *b = float(((Fl_Slider*)o)->maximum());
}
inline void fl_set_slider_xstep(Fl_Widget* o, double v) {
    ((Fl_Slider*)o)->xstep(v);
}
inline void fl_set_slider_return(Fl_Widget* o, int i) {
    ((Fl_Slider*)o)->when((Fl_When)(i|FL_WHEN_RELEASE));
}
inline void fl_set_slider_step(Fl_Widget* o, double v) {
    ((Fl_Slider *)o)->step(v);
}
inline void fl_set_slider_size(Fl_Widget* o, double v) {
    ((Fl_Slider *)o)->slider_size(v);
}

#include "Fl_Value_Slider.H"
forms_constructor(Fl_Value_Slider, fl_add_valslider)
inline void fl_set_slider_precision(Fl_Widget* o, int i) {
    ((Fl_Value_Slider*)o)->precision(i);
}
// filter function!
// The forms text object was the same as an Fl_Box except it inverted the
// meaning of FL_ALIGN_INSIDE. Implementation in forms.cxx
This file defines wrapper functions for OpenGL in FLTK.

### Functions

- **void gl_color (Fl_Color i)**
  
  Sets the current OpenGL color to an FLTK color.

- **void gl_color (int c)**
  
  Back compatibility

- **int gl_descent ()**

  Returns the current font's descent.

- **void gl_draw (const char *)**

  Draws a nul-terminated string in the current font at the current position.

- **void gl_draw (const char *, float x, float y)**

  Draws a nul-terminated string in the current font at the given position.

- **void gl_draw (const char *, int n)**

  Draws an array of n characters of the string in the current font at the current position.

- **void gl_draw (const char *, int n, float x, float y)**

  Draws n characters of the string in the current font at the current position.

- **void gl_draw (const char *, int n, int x, int y)**

  Draws n characters of the string in the current font at the given position.

- **void gl_draw (const char *, int x, int y)**

  Draws a nul-terminated string in the current font at the given position.

- **void gl_draw (const char *, int x, int y, int w, int h, Fl_Align)**

  Draws a string formatted into a box, with newlines and tabs expanded, other control characters changed to ^X.

- **void gl_draw_image (const uchar *, int x, int y, int w, int h, int d=3, int ld=0)**

- **void gl_finish ()**

  Releases an OpenGL context.

- **void gl_font (int fontid, int size)**

  Sets the current OpenGL font to the same font as calling fl_font().
- **int gl_height()**
  
  Returns the current font's height.

- **void gl_measure(const char *, int &x, int &y)**
  
  Measures how wide and tall the string will be when drawn by the gl_draw() function.

- **void gl_rect(int x, int y, int w, int h)**
  
  Outlines the given rectangle with the current color.

- **void gl_rectf(int x, int y, int w, int h)**
  
  Fills the given rectangle with the current color.

- **void gl_start()**
  
  Creates an OpenGL context.

- **int gl_texture_pile_height()**
  
  Returns the current maximum height of the pile of pre-computed string textures.

- **void gl_texture_pile_height(int max)**
  
  Changes the maximum height of the pile of pre-computed string textures.

- **void gl_texture_reset()**
  
  To call after GL operations that may invalidate textures used to draw text in GL scenes (e.g., switch between FL←DOUBLE / FL_SINGLE modes).

- **double gl_width(const char *)**
  
  Returns the width of the string in the current font.

- **double gl_width(const char *, int n)**
  
  Returns the width of n characters of the string in the current font.

- **double gl_width(uchar)**
  
  Returns the width of the character in the current font.

### 13.185.1 Detailed Description

This file defines wrapper functions for OpenGL in FLTK.

To use OpenGL from within an FLTK application you MUST use gl_visual() to select the default visual before doing show() on any windows. Mesa will crash if you try to use a visual not returned by glxChooseVisual.

Historically, this did not always work well with Fl_Double_Window's! It can try to draw into the front buffer. Depending on the system this might either crash or do nothing (when pixmaps are being used as back buffer and GL is being done by hardware), work correctly (when GL is done with software, such as Mesa), or draw into the front buffer and be erased when the buffers are swapped (when double buffer hardware is being used).

### 13.185.2 Function Documentation

#### 13.185.2.1 gl_color()

```c
void gl_color(Fl_Color i)
```

Sets the current OpenGL color to an FLTK color.

For color-index modes it will use fl_xpixel(c), which is only right if the window uses the default colormap!

#### 13.185.2.2 gl_draw() [1/7]

```c
void gl_draw(const char * str)
```

Draws a null-terminated string in the current font at the current position.

See also

- **gl_texture_pile_height(int)**
13.185.2.3  gl_draw() [2/7]
void gl_draw (
    const char * str,
    float x,
    float y )
Draws a nul-terminated string in the current font at the given position.
See also
    gl_texture_pile_height(int)

13.185.2.4  gl_draw() [3/7]
void gl_draw (    
    const char * str,
    int n )
Draws an array of n characters of the string in the current font at the current position.
See also
    gl_texture_pile_height(int)

13.185.2.5  gl_draw() [4/7]
void gl_draw (    
    const char * str,    
    int n,    
    float x,    
    float y )
Draws n characters of the string in the current font at the given position.
See also
    gl_texture_pile_height(int)

13.185.2.6  gl_draw() [5/7]
void gl_draw (    
    const char * str,    
    int n,    
    int x,    
    int y )
Draws n characters of the string in the current font at the given position.
See also
    gl_texture_pile_height(int)

13.185.2.7  gl_draw() [6/7]
void gl_draw (    
    const char * str,    
    int x,    
    int y )
Draws a nul-terminated string in the current font at the given position.
See also
    gl_texture_pile_height(int)
13.185.2.8  gl_draw() [7/7]

```c
void gl_draw (const char * str,
               int x,
               int y,
               int w,
               int h,
               Fl_Align align )
```

Draws a string formatted into a box, with newlines and tabs expanded, other control characters changed to `\X`. and aligned with the edges or center. Exactly the same output as fl_draw().

13.185.2.9  gl_font()

```c
void gl_font (int fontid,
              int size )
```

Sets the current OpenGL font to the same font as calling fl_font().

See also

```c
Fl::draw_GL_text_with_textures(int val)
```

13.185.2.10  gl_rect()

```c
void gl_rect (int x,
              int y,
              int w,
              int h )
```

Outlines the given rectangle with the current color.
If Fl_Gl_Window::ortho() has been called, then the rectangle will exactly fill the given pixel rectangle.

13.185.2.11  gl_rectf()

```c
void gl_rectf (int x,
               int y,
               int w,
               int h )
```

Fills the given rectangle with the current color.

See also

```c
gl_rect(int x, int y, int w, int h)
```

13.185.2.12  gl_texture_pile_height() [1/2]

```c
int gl_texture_pile_height ( void )
```

Returns the current maximum height of the pile of pre-computed string textures.
The default value is 100

See also

```c
Fl::draw_GL_text_with_textures(int)
```
void gl_texture_pile_height (  
    int max  )

Changes the maximum height of the pile of pre-computed string textures.
Strings that are often re-displayed can be processed much faster if this pile is set high enough to hold all of them.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>Maximum height of the texture pile</td>
</tr>
</tbody>
</table>

See also

Fl::draw_GL_text_with_textures(int)

Go to the documentation of this file.
13.187  gl2openGl.h

13.188  gl_draw.H
# glu.h

#include "gl.h"

extern FL_EXPORT void gl_remove_displaylist_fonts();

---

# glut.H

#include "Enumerations.H" // for color names
#ifndef __WIN32
#define __WIN32
#endif
#include <windows.h>
#endif

#ifndef APIENTRY
#define APIENTRY __attribute__ ((__stdcall__))
#endif

#ifdef __APPLE__ // PORTME: OpenGL Path abstraction
#include <OpenGL/glu.h>
#else
#include <GL/glu.h>
#endif
#endif // !FL_glu_H

---
// Notes: as pointed out in STR #3458 the current GLUT window,
// i.e. the global static variable 'glut_window' can be NULL ...
// (a) if not (yet) initialized
// (b) if the current GLUT window is deleted at any time.
// The FLTK implementation silently ignores function calls if the current
// window is NULL to avoid dereferencing a NULL pointer. This is obviously
// compatible with GLUT version 3.7 according to comment #5 on STR #3458.
// According to the same comment FreeGLUT 3.0 would issue an error message
// and quit.
// Albrecht-S, Oct 2023

#ifndef _FL_glut_H_
#define _FL_glut_H_

#include "gl.h"

#include "Fl.H"
#include "Fl_Gl_Window.H"

class FL_EXPORT Fl_Glut_Window : public Fl_Gl_Window {
    void _init();
    int mouse_down;
protected:
    int mouse_down;
private:
    void draw() FL_OVERRIDE;
    int mouse_down;
    int (mouse)(int b, int state, int x, int y);
public:
    Fl_Glut_Window(int w, int h, const char *t=0);
    Fl_Glut_Window(int x, int y, int w, int h, const char *t=0);
    ~Fl_Glut_Window();
    void* make_current();
    void* (draw_overlay)();
    void* (draw)(void); (display)();
    void* (reshape)(int w, int h);
    void* (keyboard)(uchar, int x, int y);
    void* (mouse)(int b, int state, int x, int y);
    void* (motion)(int x, int y);
    void* (passivemotion)(int x, int y);
    void* (entry)(int);
    void* (visibility)(int);
    void* (special)(int, int x, int y);
    void* (handle)(int) FL_OVERRIDE;
    int handle(int) FL_OVERRIDE;
    int number;
    int menu[3];
    int (number);
public:
    void make_current();
    void (draw_overlay)();
    void (draw)(); (display)();
    void (reshape)(int w, int h);
    void (keyboard)(uchar, int x, int y);
    void (mouse)(int b, int state, int x, int y);
    void (motion)(int x, int y);
    void (passivemotion)(int x, int y);
    void (entry)(int);
    void (visibility)(int);
    void (special)(int, int x, int y);
    Fl_Glut_Window(int w, int h, const char *t=0);
    Fl_Glut_Window(int x, int y, int w, int h, const char *t=0);
    ~Fl_Glut_Window();
    Fl_Glut_Window();
};

extern FL_EXPORT Fl_Glut_Window *glut_window; // the current window
extern FL_EXPORT Fl_GL_Window* glut_window; // the current window
extern FL_EXPORT int glut_menu; // the current menu
extern FL_EXPORT int glut_menu; // the current menu

// function pointers that are not per-window:
extern FL_EXPORT void (*glut_idle_function)();
extern FL_EXPORT void (*glut_menustate_function)(int);
extern FL_EXPORT void (*glut_menustatus_function)(int, int, int);

#define GLUT_API_VERSION This does not match any version of GLUT exactly...

FL_EXPORT void glutInit(int *argcp, char **argv); // creates first window
FL_EXPORT void glutInitDisplayMode(unsigned int mode);

// the FL symbols have the same value as the GLUT ones:
#define GL_RGB FL_RGB
#define GLUT_RGB FL_RGB
#define GL_INDEX FL_INDEX
#define GLUT_INDEX FL_INDEX
#define GLUT_RGBA FL_RGBA
#define GL_RGBA FL_RGBA
#define GLUT_DOUBLE FL_DOUBLE
#define GL_DOUBLE FL_DOUBLE
#define GLUT_INDEX FL_INDEX
#define GLUT_INDEX FL_INDEX
#define GLUT_ALPHA FL_ALPHA
#define GL_ALPHA FL_ALPHA
#define GLUT_DEPTH FL_DEPTH
#define GL_DEPTH FL_DEPTH
#define GLUT_STENCIL FL_STENCIL
#define GLUT_STENCIL FL_STENCIL
#define GLUT_MULTISAMPLE FL_MULTISAMPLE
#define GLUT_MULTISAMPLE FL_MULTISAMPLE
#define GLUT_SERRER FL_SERRER
#define GLUT_SERRER FL_SERRER
#define GLUT_LUMINANCE 512
#define GLUT_LUMINANCE 512

FL_EXPORT void glutInitWindowSize(int w, int h);

FL_EXPORT void glutInitWindowPosition(int x, int y);

FL_EXPORT void glutCreateWindow(char *title);
FL_EXPORT void glutCreateWindow(const char *title);

FL_EXPORT int glutCreateSubWindow(int win, int x, int y, int width, int height);

FL_EXPORT void glutDestroyWindow(int win);
inline void glutPostRedisplay() {
    if (glut_window) glut_window->redraw();
}

inline int glutGetWindow() {
    return glut_window ? glut_window->number : 0;
}

inline void glutSetWindowTitle(char *t) {
    if (glut_window) glut_window->label(t);
}

inline void glutSetIconTitle(char *t) {
    if (glut_window) glut_window->iconlabel(t);
}

inline void glutPositionWindow(int x, int y) {
    if (glut_window) glut_window->position(x,y);
}

inline void glutReshapeWindow(int w, int h) {
    if (glut_window) glut_window->size(w,h);
}

inline void glutPopWindow() {
    if (glut_window) glut_window->show();
}

inline void glutPushWindow() { /* do nothing */ }

inline void glutIconifyWindow() {
    if (glut_window) glut_window->iconize();
}

inline void glutShowWindow() {
    if (glut_window) glut_window->show();
}

inline void glutHideWindow() {
    if (glut_window) glut_window->hide();
}

inline void glutFullScreen() {
    if (glut_window) glut_window->fullscreen();
}

inline void glutSetCursor(Fl_Cursor cursor) {
    if (glut_window) glut_window->cursor(cursor);
}

#define GLUT_CURSOR_RIGHT_ARROW ((Fl_Cursor)2)
#define GLUT_CURSOR_LEFT_ARROW ((Fl_Cursor)67)
#define GLUT_CURSOR_INFO FL_CURSOR_HAND
#define GLUT_CURSOR_DESTROY ((Fl_Cursor)45)
#define GLUT_CURSOR_HELP FL_CURSOR_HELP
#define GLUT_CURSOR_CYCLE ((Fl_Cursor)26)
#define GLUT_CURSOR_SPRAY ((Fl_Cursor)63)
#define GLUT_CURSOR_WAIT FL_CURSOR_WAIT
#define GLUT_CURSOR_TEXT FL_CURSOR_INSERT
#define GLUT_CURSOR_CROSSHAIR FL_CURSOR_CROSS
#define GLUT_CURSOR_UP_DOWN FL_CURSOR_NS
#define GLUT_CURSOR_LEFT_RIGHT FL_CURSOR_WE
#define GLUT_CURSOR_TOP_SIDE FL_CURSOR_N
#define GLUT_CURSOR_BOTTOM_SIDE FL_CURSOR_S
#define GLUT_CURSOR_LEFT_SIDE FL_CURSOR_W
#define GLUT_CURSOR_RIGHT_SIDE FL_CURSOR_E
#define GLUT_CURSOR_TOP_LEFT_CORNER FL_CURSOR_NW
#define GLUT_CURSOR_TOP_RIGHT_CORNER FL_CURSOR_NE
#define GLUT_CURSOR_BOTTOM_RIGHT_CORNER FL_CURSOR_SE
#define GLUT_CURSOR_BOTTOM_LEFT_CORNER FL_CURSOR_SW
#define GLUT_CURSOR_INHERIT FL_CURSOR_DEFAULT
#define GLUT_CURSOR_NONE FL_CURSOR_NONE
#define GLUT_CURSOR_FULL_CROSSHAIR FL_CURSOR_CROSS

inline void glutWarpPointer(int, int) { /* do nothing */ }

inline void glutEstablishOverlay() {
    if (glut_window) glut_window->make_overlay_current();
}

// notice that the numeric values are different than glut:
#define GLUT_CURSOR_RIGHT_ARROW (2)
#define GLUT_CURSOR_LEFT_ARROW (67)
#define GLUT_CURSOR_INFO FL_CURSOR_HAND
#define GLUT_CURSOR_DESTROY (45)
#define GLUT_CURSOR_HELP FL_CURSOR_HELP
#define GLUT_CURSOR_CYCLE (26)
#define GLUT_CURSOR_SPRAY (63)
#define GLUT_CURSOR_WAIT FL_CURSOR_WAIT
#define GLUT_CURSOR_TEXT FL_CURSOR_INSERT
#define GLUT_CURSOR_CROSSHAIR FL_CURSOR_CROSS
#define GLUT_CURSOR_UP_DOWN FL_CURSOR_NS
#define GLUT_CURSOR_LEFT_RIGHT FL_CURSOR_WE
#define GLUT_CURSOR_TOP_SIDE FL_CURSOR_N
#define GLUT_CURSOR_BOTTOM_SIDE FL_CURSOR_S
#define GLUT_CURSOR_LEFT_SIDE FL_CURSOR_W
#define GLUT_CURSOR_RIGHT_SIDE FL_CURSOR_E
#define GLUT_CURSOR_TOP_LEFT_CORNER FL_CURSOR_NW
#define GLUT_CURSOR_TOP_RIGHT_CORNER FL_CURSOR_NE
#define GLUT_CURSOR_BOTTOM_RIGHT_CORNER FL_CURSOR_SE
#define GLUT_CURSOR_BOTTOM_LEFT_CORNER FL_CURSOR_SW
#define GLUT_CURSOR_INHERIT FL_CURSOR_DEFAULT
#define GLUT_CURSOR_NONE FL_CURSOR_NONE
#define GLUT_CURSOR_FULL_CROSSHAIR FL_CURSOR_CROSS

// notice that the numeric values are different than glut:
#define GLUT_CURSOR_RIGHT_ARROW (2)
#define GLUT_CURSOR_LEFT_ARROW (67)
#define GLUT_CURSOR_INFO FL_CURSOR_HAND
#define GLUT_CURSOR_DESTROY (45)
#define GLUT_CURSOR_HELP FL_CURSOR_HELP
#define GLUT_CURSOR_CYCLE (26)
#define GLUT_CURSOR_SPRAY (63)
#define GLUT_CURSOR_WAIT FL_CURSOR_WAIT
#define GLUT_CURSOR_TEXT FL_CURSOR_INSERT
#define GLUT_CURSOR_CROSSHAIR FL_CURSOR_CROSS
#define GLUT_CURSOR_UP_DOWN FL_CURSOR_NS
#define GLUT_CURSOR_LEFT_RIGHT FL_CURSOR_WE
#define GLUT_CURSOR_TOP_SIDE FL_CURSOR_N
#define GLUT_CURSOR_BOTTOM_SIDE FL_CURSOR_S
#define GLUT_CURSOR_LEFT_SIDE FL_CURSOR_W
#define GLUT_CURSOR_RIGHT_SIDE FL_CURSOR_E
#define GLUT_CURSOR_TOP_LEFT_CORNER FL_CURSOR_NW
#define GLUT_CURSOR_TOP_RIGHT_CORNER FL_CURSOR_NE
#define GLUT_CURSOR_BOTTOM_RIGHT_CORNER FL_CURSOR_SE
#define GLUT_CURSOR_BOTTOM_LEFT_CORNER FL_CURSOR_SW
#define GLUT_CURSOR_INHERIT FL_CURSOR_DEFAULT
#define GLUT_CURSOR_NONE FL_CURSOR_NONE
#define GLUT_CURSOR_FULL_CROSSHAIR FL_CURSOR_CROSS

// notice that the numeric values are different than glut:
#define GLUT_CURSOR_RIGHT_ARROW (2)
#define GLUT_CURSOR_LEFT_ARROW (67)
#define GLUT_CURSOR_INFO FL_CURSOR_HAND
#define GLUT_CURSOR_DESTROY (45)
#define GLUT_CURSOR_HELP FL_CURSOR_HELP
#define GLUT_CURSOR_CYCLE (26)
#define GLUT_CURSOR_SPRAY (63)
#define GLUT_CURSOR_WAIT FL_CURSOR_WAIT
#define GLUT_CURSOR_TEXT FL_CURSOR_INSERT
#define GLUT_CURSOR_CROSSHAIR FL_CURSOR_CROSS
#define GLUT_CURSOR_UP_DOWN FL_CURSOR_NS
#define GLUT_CURSOR_LEFT_RIGHT FL_CURSOR_WE
#define GLUT_CURSOR_TOP_SIDE FL_CURSOR_N
#define GLUT_CURSOR_BOTTOM_SIDE FL_CURSOR_S
#define GLUT_CURSOR_LEFT_SIDE FL_CURSOR_W
#define GLUT_CURSOR_RIGHT_SIDE FL_CURSOR_E
#define GLUT_CURSOR_TOP_LEFT_CORNER FL_CURSOR_NW
#define GLUT_CURSOR_TOP_RIGHT_CORNER FL_CURSOR_NE
#define GLUT_CURSOR_BOTTOM_RIGHT_CORNER FL_CURSOR_SE
#define GLUT_CURSOR_BOTTOM_LEFT_CORNER FL_CURSOR_SW
#define GLUT_CURSOR_INHERIT FL_CURSOR_DEFAULT
#define GLUT_CURSOR_NONE FL_CURSOR_NONE
#define GLUT_CURSOR_FULL_CROSSHAIR FL_CURSOR_CROSS
inline void glutRemoveOverlay() {
    if (glut_window) glut_window->hide_overlay();
}

inline void glutUseLayer(GLenum layer) {
    if (!glut_window) return;
    layer ? glut_window->make_overlay_current() : glut_window->make_current();
}

enum {GLUT_NORMAL, GLUT_OVERLAY};

inline void glutPostOverlayRedisplay() {
    if (glut_window) glut_window->redraw_overlay();
}

inline void glutShowOverlay() {
    if (glut_window) glut_window->redraw_overlay();
}

inline void glutHideOverlay() {
    if (glut_window) glut_window->hide_overlay();
}

inline void glutAttachMenu(int b) {
    if (glut_window) glut_window->menu[b] = glut_menu;
}

inline void glutDetachMenu(int b) {
    if (glut_window) glut_window->menu[b] = 0;
}

enum {GLUT_LEFT, GLUT_ENTERED};

inline void glutEntryFunc(void (*f)(int s)) {
    if (glut_window) glut_window->entry = f;
}

# define GLUT_LEFT_BUTTON 0
# define GLUT_MIDDLE_BUTTON 1
# define GLUT_RIGHT_BUTTON 2
# define GLUT_DOWN 0
# define GLUT_UP 1

inline void glutMotionFunc(void (*f)(int x, int y)) {
    if (glut_window) glut_window->motion = f;
}

inline void glutPassiveMotionFunc(void (*f)(int x, int y)) {
    if (glut_window) glut_window->passivemotion = f;
}

inline void glutEntryFunc(void (*f)(int s)) {
    if (glut_window) glut_window->entry = f;
}

enum {GLUT_LEFT, GLUT_ENTERED};

inline void glutEntryFunc(void (*f)(int s)) {
if (glut_window) glut_window->visibility = f;
enum {GLUT_NOT_VISIBLE, GLUT_VISIBLE};

FL_EXPORT void glutIdleFunc(void (*)(void *));
inline void glutTimerFunc(unsigned int msec, void (*)(int), int value) {
  Fl::add_timeout((msec * 0.001), (void (*)(void *))f, (void *)(fl_intptr_t)value);
}
inline void glutMenuStateFunc(void (*)(int state)) {
  glut_menustate_function = f;
}
inline void glutMenuStatusFunc(void (*)(int status, int x, int y)) {
  glut_menustatus_function = f;
}
enum {GLUT_MENU_NOT_IN_USE, GLUT_MENU_IN_USE};
inline void glutSpecialFunc(void (*)(int key, int x, int y)) {
  if (glut_window) glut_window->special = f;
}

#define GLUT_KEY_F1 1
#define GLUT_KEY_F2 2
#define GLUT_KEY_F3 3
#define GLUT_KEY_F4 4
#define GLUT_KEY_F5 5
#define GLUT_KEY_F6 6
#define GLUT_KEY_F7 7
#define GLUT_KEY_F8 8
#define GLUT_KEY_F9 9
#define GLUT_KEY_F10 10
#define GLUT_KEY_F11 11
#define GLUT_KEY_F12 12
// WARNING: Different values than GLUT uses:
#define GLUT_KEY_LEFT FL_Left
#define GLUT_KEY_UP FL_Up
#define GLUT_KEY_RIGHT FL_Right
#define GLUT_KEY_DOWN FL_Down
#define GLUT_KEY_PAGE_UP FL_Page_Up
#define GLUT_KEY_PAGE_DOWN FL_Page_Down
#define GLUT_KEY_HOME FL_Home
#define GLUT_KEY_END FL_End
#define GLUT_KEY_INSERT FL_Insert

// inline void glutSpaceballMotionFunc(void (*)(int x, int y, int z));
// inline void glutSpaceballRotateFunc(void (*)(int x, int y, int z));
// inline void glutSpaceballButtonFunc(void (*)(int button, int state));
// inline void glutButtonBoxFunc(void (*)(int button, int state));
// inline void glutDialsFunc(void (*)(int dial, int value));
// inline void glutTabletMotionFunc(void (*)(int x, int y));
// inline void glutTabletButtonFunc(void (*)(int button, int state, int x, int y));
inline void glutOverlayDisplayFunc(void (*)(void *)) {
  if (glut_window) glut_window->overlaydisplay = f;
}

// inline void glutWindowStatusFunc(void (*)(int state));
enum {GLUT_HIDDEN, GLUT_FULLY_RETAINTED, GLUT_PARITALLY_RETAINTED,
      GLUT_FULLY_COVERED};
inline void glutSetColor(int ndx, GLfloat red, GLfloat green, GLfloat blue);
inline GLfloat glutGetColor(int ndx, int component);
#define GLUT_RED 0
#define GLUT_GREEN 1
#define GLUT_BLUE 2
// inline void glutCopyColormap(int win);
// Warning: values are changed from GLUT!
// Also relies on the GL_ symbols having values greater than 100
FL_EXPORT int glutGet(GLenum type);
enum {
  GLUT_RETURN_ZERO = 0,
  GLUT_WINDOW_X,
  GLUT_WINDOW_Y,
  GLUT_WINDOW_WIDTH,
  GLUT_WINDOW_HEIGHT,
  GLUT_WINDOW_PARENT,
  GLUT_SCREEN_WIDTH,
  GLUT_SCREEN_HEIGHT,
# define GLUT_WINDOW_STENCIL_SIZE GL_STENCIL_BITS
# define GLUT_WINDOW_DEPTH_SIZE GL_DEPTH_BITS
# define GLUT_WINDOW_GREEN_SIZE GL_GREEN_BITS
# define GLUT_WINDOW_RED_SIZE GL_RED_BITS
# define GLUT_WINDOW_ALPHA_SIZE GL_ALPHA_BITS
# define GLUT_WINDOW_ACCUM_RED_SIZE GL_ACCUM_RED_BITS
# define GLUT_WINDOW_ACCUM_GREEN_SIZE GL_ACCUM_GREEN_BITS
# define GLUT_WINDOW_ACCUM_BLUE_SIZE GL_ACCUM_BLUE_BITS
# define GLUT_WINDOW_ACCUM_ALPHA_SIZE GL_ACCUM_ALPHA_BITS
# define GLUT_WINDOW_DOUBLEBUFFER GL_DOUBLEBUFFER
# define GLUT_WINDOW_RGBA GL_RGBA
# define GLUT_WINDOW_COLORMAP_SIZE GL_INDEX_BITS
# ifdef GL_SAMPLES_SGIS
# define GLUT_WINDOW_NUM_SAMPLES GL_SAMPLES_SGIS
# else
# define GLUT_WINDOW_NUM_SAMPLES GLUT_RETURN_ZERO
# endif
# define GLUT_WINDOW_STEREO GL_STEREO

// WARNING: these values are different than GLUT uses:
# define GLUT_ACTIVE_SHIFT FL_SHIFT
# define GLUT_ACTIVE_CTRL FL_CTRL

inline int glutGetModifiers() {
    return Fl::event_state() & (GLUT_ACTIVE_SHIFT | GLUT_ACTIVE_CTRL | GLUT_ACTIVE_ALT);
}

FL_EXPORT int glutDeviceGet(GLenum type);

extern "C" {
    typedef void (*GLUTproc)();
}

FL_EXPORT int glutLayerGet(GLenum);

// inline int glutVideoResizeGet(GLenum param);
// #define GLUT_VIDEO_RESIZE_POSSIBLE 900
// #define GLUT_VIDEO_RESIZE_IN_USE 901
// #define GLUT_VIDEO_RESIZE_WIDTH_DELTA 902
// #define GLUT_VIDEO_RESIZE_HEIGHT_DELTA 903
// #define GLUT_VIDEO_RESIZE_WIDTH 904
// #define GLUT_VIDEO_RESIZE_HEIGHT 905

// inline void glutSetupVideoResizing();
// inline void glutStopVideoResizing();
inline void glutVideoResize(int x, int y, int width, int height);
inline void glutVideoPan(int x, int y, int width, int height);

Font argument must be a void* for compatibility, so...
struct FL_Glut_Bitmap_Font {FL_Font font; FL_Fontsize size;};

inline void glutVideoResize(int x, int y, int width, int height);
inline void glutVideoPan(int x, int y, int width, int height);

Font argument must be a void* for compatibility, so...
struct FL_Glut_Bitmap_Font {FL_Font font; FL_Fontsize size;};

extern FL_EXPORT struct Fl_Glut_Bitmap_Font glutBitmap9By15, glutBitmap8By13, glutBitmapTimesRoman10,
                glutBitmapTimesRoman24, glutBitmapHelvetica10, glutBitmapHelvetica12,
                glutBitmapHelvetica18;

#define GLUT_BITMAP_9_BY_15 (&glutBitmap9By15)
#define GLUT_BITMAP_8_BY_13 (&glutBitmap8By13)
#define GLUT_BITMAP_TIMES_ROMAN_10 (&glutBitmapTimesRoman10)
#define GLUT_BITMAP_TIMES_ROMAN_24 (&glutBitmapTimesRoman24)
#define GLUT_BITMAP_HELVETICA_10 (&glutBitmapHelvetica10)
#define GLUT_BITMAP_HELVETICA_12 (&glutBitmapHelvetica12)
#define GLUT_BITMAP_HELVETICA_18 (&glutBitmapHelvetica18)

FL_EXPORT void glutBitmapCharacter(void *font, int character);
FL_EXPORT int glutBitmapHeight(void *font);
FL_EXPORT int glutBitmapLength(void *font, const unsigned char *string);
FL_EXPORT void glutBitmapString(void *font, const unsigned char *string);
FL_EXPORT int glutBitmapWidth(void *font, int character);

FL_EXPORT int glutExtensionSupported(char *name);

/* GLUT stroked font sub-API */
struct Fl_Glut_StrokeVertex {
  GLfloat X, Y;
};
struct Fl_Glut_StrokeStrip {
  int Number;
  const Fl_Glut_StrokeVertex * Vertices;
};
struct Fl_Glut_StrokeChar {
  GLfloat Right;
  int Number;
  const Fl_Glut_StrokeStrip * Strips;
};
struct Fl_Glut_StrokeFont {
  char * Name; // The source font name
  int Quantity; // Number of chars in font
  GLfloat Height; // Height of the characters
  const Fl_Glut_StrokeChar ** Characters; // The characters mapping
};

extern FL_EXPORT Fl_Glut_StrokeFont glutStrokeRoman;
extern FL_EXPORT Fl_Glut_StrokeFont glutStrokeMonoRoman;
#define GLUT_STROKE_ROMAN (&glutStrokeRoman)
#define GLUT_STROKE_MONO_ROMAN (&glutStrokeMonoRoman)

FL_EXPORT void glutStrokeCharacter(void *font, int character);
FL_EXPORT GLfloat glutStrokeHeight(void *font);
FL_EXPORT int glutStrokeLength(void *font, const unsigned char *string);
FL_EXPORT void glutStrokeString(void *font, const unsigned char *string);
FL_EXPORT int glutStrokeWidth(void *font, int character);

/* GLUT pre-built models sub-API */
FL_EXPORT void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);
FL_EXPORT void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
FL_EXPORT void glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);
FL_EXPORT void glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);
FL_EXPORT void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);
FL_EXPORT void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);
FL_EXPORT void glutWireDodecahedron();
FL_EXPORT void glutSolidDodecahedron();
FL_EXPORT void glutWireTeapot(GLdouble size);
FL_EXPORT void glutSolidTeapot(GLdouble size);
FL_EXPORT void glutWireOctahedron();
FL_EXPORT void glutSolidOctahedron();
FL_EXPORT void glutWireTetrahedron();
FL_EXPORT void glutSolidTetrahedron();
FL_EXPORT void glutWireIcosahedron();
FL_EXPORT void glutSolidIcosahedron();

#endif // !_FL_glut_H_
13.191 mac.H File Reference

Mac OS X-specific symbols.

Classes

• class Fl_Mac_App_Menu

Functions

• Fl_Window * fl_mac_find (FLWindow *)
  Returns the Fl_Window corresponding to the given macOS-specific window reference.

• CGContextRef fl_mac_gc ()
  Returns the macOS-specific graphics context for the current window.

• void fl_mac_set_about (Fl_Callback *cb, void *user_data, int shortcut=0)
  Attaches a callback to the "About myprog" item of the system application menu.

• FLWindow * fl_mac_xid (const Fl_Window *win)
  Returns the macOS-specific window reference corresponding to the given Fl_Window object.

Variables

• int fl_mac_os_version
  The version number of the running Mac OS X (e.g., 100604 for 10.6.4, 101300 for 10.13, 140102 for 14.1.2).

13.191.1 Detailed Description

Mac OS X-specific symbols.

13.192 mac.H

Go to the documentation of this file.
typedef class FLWindow *Window; // pointer to the FLWindow objective-c class

#include <FL/Fl_Widget.H> // for Fl_Callback

#ifdef (defined(FL_LIBRARY) || defined(FL_INTERNALS)) // this part must be compiled when building the FLTK libraries

#include <ApplicationServices/ApplicationServices.h>

#undef check // because of Fl::check()

#ifndef MAC_OS_X_VERSION_10_4
#define MAC_OS_X_VERSION_10_4 1040
#endif

#ifndef MAC_OS_X_VERSION_10_5
#define MAC_OS_X_VERSION_10_5 1050
#endif

#ifndef MAC_OS_X_VERSION_10_6
#define MAC_OS_X_VERSION_10_6 1060
#endif

#ifndef MAC_OS_X_VERSION_10_7
#define MAC_OS_X_VERSION_10_7 1070
#endif

#ifndef MAC_OS_X_VERSION_10_8
#define MAC_OS_X_VERSION_10_8 1080
#endif

#ifndef MAC_OS_X_VERSION_10_9
#define MAC_OS_X_VERSION_10_9 1090
#endif

#ifndef MAC_OS_X_VERSION_10_10
#define MAC_OS_X_VERSION_10_10 101000
#endif

#ifndef MAC_OS_X_VERSION_10_11
#define MAC_OS_X_VERSION_10_11 101100
#endif

#ifndef MAC_OS_X_VERSION_10_12
#define MAC_OS_X_VERSION_10_12 101200
#endif

#ifndef MAC_OS_X_VERSION_10_13
#define MAC_OS_X_VERSION_10_13 101300
#endif

#ifndef MAC_OS_X_VERSION_10_14
#define MAC_OS_X_VERSION_10_14 101400
#endif

#ifndef MAC_OS_X_VERSION_10_15
#define MAC_OS_X_VERSION_10_15 101500
#endif

#ifndef MAC_OS_X_VERSION_10_16
#define MAC_OS_X_VERSION_10_16 101600
#endif

#ifndef MAC_OS_VERSION_11_0
#define MAC_OS_VERSION_11_0 110000
#endif

#ifndef MAC_OS_VERSION_12_0
#define MAC_OS_VERSION_12_0 120000
#endif

#ifndef MAC_OS_VERSION_13_0
#define MAC_OS_VERSION_13_0 130000
#endif

#ifndef MAC_OS_VERSION_14_0
#define MAC_OS_VERSION_14_0 140000
#endif

struct flCocoaRegion {
  int count;
  CGRect *rects;
}; // a region is the union of a series of rectangles

typedef(CharSequence) CFString; // appears with 10.5 in CMCoreText.h

if defined(_LP64__) && __LP64__
  typedef long NSInteger;
  typedef unsigned long NSUInteger;
else
  typedef int NSInteger;
  typedef unsigned int NSUInteger;
endif

if MAC_OS_X_VERSION_MAX_ALLOWED < MAC_OS_X_VERSION_10_4
  typedef CGImageAlphaInfo CGBitmapInfo;
endif

struct flCocoaRegion {
  int count;
  CGRect *rects;
}; // a region is the union of a series of rectangles

if defined(_LP64__) && __LP64__
  typedef double CGFloat;
else
  typedef double CGFloat;
endif

Generated by Doxygen
13.193  math.h

// Math header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2020 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#define M_PI 3.14159265358979323846
#define M_PI_2 1.57079632679489661923
#define M_PI_4 0.78539816339744830962
#define M_1_PI 0.31830988618379067154
#define M_2_PI 0.63661977236758134308

#define M_SQRT2 1.41421356237309504880
#define M_SQRT1_2 0.70710678118654752440

#ifndef _WIN32 || _ CRAY || __MINGW32__
This file defines arrays of human readable names for FLTK symbolic constants.

### Variables

- `const char * const fl_callback_reason_names []`
  
  This is an array of callback reason names you can use to convert font numbers into names.

- `const char * const fl_eventnames []`
  
  This is an array of event names you can use to convert event numbers into names.

- `const char * const fl_fontnames []`
  
  This is an array of font names you can use to convert font numbers into names.

### Detailed Description

This file defines arrays of human readable names for FLTK symbolic constants.

---

Go to the documentation of this file.

This library is free software. Distribution and use rights are outlined in the file "COPYING" which should have been included with this file. If this file is missing or damaged, see the license at:

- [https://www.fltk.org/COPYING.php](https://www.fltk.org/COPYING.php)
- [https://www.fltk.org/bugs.php](https://www.fltk.org/bugs.php)

Thanks to Greg Ercolano for this addition.
const char * const fl_fontnames[] =
{
  "FL_HELVETICA",
  "FL_HELVETICA_BOLD",
  "FL_HELVETICA_ITALIC",
  "FL_HELVETICA_BOLD_ITALIC",
  "FL_COURIER",
  "FL_COURIER_BOLD",
  "FL_COURIER_ITALIC",
  "FL_COURIER_BOLD_ITALIC",
  "FL_TIMES",
  "FL_TIMES_BOLD",
  "FL_TIMES_ITALIC",
  "FL_TIMES_BOLD_ITALIC",
  "FL_SYMBOL",
  "FL_SCREEN",
  "FL_SCREEN_BOLD",
  "FL_ZAPF_DINGBATS",
};

const char * const fl_callback_reason_names[] =
{
  "FL_REASON_UNKNOWN",
  "FL_REASON_SELECTED",
  "FL_REASON_DESELECTED",
  "FL_REASON_RESELECTED",
  "FL_REASON_OPENED",
  "FL_REASON_CLOSED",
  "FL_REASON_DRAGGED",
  "FL_REASON_CANCELLED",
  "FL_REASON_CHANGED",
  "FL_REASON_GOT_FOCUS",
  "FL_REASON_LOST_FOCUS",
  "FL_REASON_RELEASED",
  "FL_REASON_USER",
  "FL_REASON_USER+1",
  "FL_REASON_USER+2",
  "FL_REASON_USER+3",
};

#endif /* FL_NAMES_H */
Definitions of platform-dependent types.

**Macros**

- `#define FL_COMMAND opaque
  An alias for FL_CTRL on Windows and X11, or FL_META on MacOS X.

- `#define FL_CONTROL opaque
  An alias for FL_META on Windows and X11, or FL_CTRL on MacOS X.

**Typedefs**

- `typedef opaque fl_intptr_t
  An integral type large enough to store a pointer or a long value.

- `typedef opaque Fl_Offscreen
  Platform-specific value representing an offscreen drawing buffer.

- `typedef struct opaque Fl_Region
  Pointer to a platform-specific structure representing a collection of rectangles.

- `typedef opaque FL_SOCKET

Generated by Doxygen
socket or file descriptor

- typedef opaque Fl_Timestamp
  Platform-specific point in time, used for delta time calculation.
- typedef opaque fl_uintptr_t
  An unsigned integral type large enough to store a pointer or an unsigned long value.
- typedef struct opaque * GLContext
  Pointer to a platform-specific structure representing the window's OpenGL rendering context.

### 13.197.1 Detailed Description

Definitions of platform-dependent types.

The exact nature of these types varies with the platform. Therefore, portable FLTK applications should not assume these types have a specific size, or that they are pointers.

### 13.197.2 Typedef Documentation

#### 13.197.2.1 fl_intptr_t

typedef opaque fl_intptr_t

An integral type large enough to store a pointer or a long value.

A pointer value can be safely cast to fl_intptr_t, and later cast back to its initial pointer type without change to the pointer value. A variable of type fl_intptr_t can also store a long int value.

#### 13.197.2.2 Fl_Offscreen

typedef opaque Fl_Offscreen

Platform-specific value representing an offscreen drawing buffer.

Note

This value can be safely cast to these types on each platform:

- X11: Pixmap
- Wayland: cairo_t *
- Windows: HBITMAP
- macOS: CGContextRef

#### 13.197.2.3 Fl_Region

typedef struct opaque* Fl_Region

Pointer to a platform-specific structure representing a collection of rectangles.

Note

This pointer can be safely cast to these types on each platform:

- X11: Region as defined by X11
- Wayland: struct flCairoRegion *
- Windows: HRGN
- macOS: struct flCocoaRegion *

#### 13.197.2.4 Fl_Timestamp

typedef opaque Fl_Timestamp

Platform-specific point in time, used for delta time calculation.

Note

This type may be a struct. sizeof(Fl_Timestamp) may be different on different platforms. Fl_Timestamp may change with future ABI changes.
**fl guintptr_t**

typedef opaque fl guintptr_t

An unsigned integral type large enough to store a pointer or an unsigned long value. A pointer value can be safely cast to fl guintptr_t, and later cast back to its initial pointer type without change to the pointer value. A variable of type fl guintptr_t can also store an unsigned long int value.

**GLContext**

typedef struct opaque GLContext

Pointer to a platform-specific structure representing the window's OpenGL rendering context.

**Note**

This pointer can be safely cast to these types on each platform:

- X11: GLXContext
- Wayland: EGLContext
- Windows: HGLRC
- macOS: NSOpenGLContext

---

Go to the documentation of this file.

---

Go to the documentation of this file.
#ifdef _WIN64
#include <stddef.h> /* stdint.h not available before VS 2010 (1600) */
#else
#include <stdint.h>
#endif

typedef intptr_t fl_intptr_t;
typedef uintptr_t fl_uintptr_t;

#else /* ! _WIN64 */
typedef long fl_intptr_t;
typedef unsigned long fl_uintptr_t;
#endif /* _WIN64 */

typedef void *GLContext;
typedef void *Fl_Region;
typedef fl_uintptr_t Fl_Offscreen;

/* Allows all hybrid combinations except WIN32 + X11 with MSVC */
#if defined(_WIN32) && !defined(__MINGW32__)
struct dirent {char d_name[1];};
#else
#include <dirent.h>
#endif

#define FL_COMMAND fl_command_modifier()
#define FL_CONTROL fl_control_modifier()

#endif /* FL_PLATFORM_TYPES_H */

// This is currently the same for all platforms, but may change in the future
struct Fl_Timestamp_t {
time_t sec;
int usec;
};

typedef struct Fl_Timestamp_t Fl_Timestamp;

*/ 

13.199 wayland.H File Reference

Definitions of functions specific to the Wayland platform.

Typedefs

- typedef struct _cairo cairo_t
- typedef void * EGLContext

Functions

- struct wl_compositor * fl_wl_compositor ()
  Returns the wl_compositor of the current Wayland session.
- struct wl_display * fl_wl_display ()
  Returns the Wayland display in use.
- Fl_Window * fl_wl_find (struct wl_window *)
  Returns the Fl_Window corresponding to a given the platform-specific window reference.
- cairo_t * fl_wl_gc ()
  Returns the cairo context associated to the current window or Fl_Image_Surface.
Definitions of functions specific to the Wayland platform.

13.199.2 Function Documentation

13.199.2.1 fl_wl_compositor()

```
struct wl_compositor * fl_wl_compositor ( ) [extern]
```

Returns the wl_compositor of the current Wayland session. This allows, for example, to create a wl_surface with
```
struct wl_surface *my_wl_surface = wl_compositor_create_surface(fl_wl_compositor());
```

13.200 wayland.H

Go to the documentation of this file.

Definitions of functions specific to the Windows platform.

13.201 win32.H File Reference

Definitions of functions specific to the Windows platform.
Functions

- **HINSTANCE fl_win32_display ()**
  
  Returns the Windows-specific display in use

- **FL_Window * fl_win32_find (HWND)**
  
  Returns the *FL_Window* corresponding to the given Windows-specific window reference.

- **HDC fl_win32_gc ()**
  
  Returns the Windows-specific graphics context for the current window

- **HGLRC fl_win32_glcontext (GLContext rc)**
  
  Returns the Windows-specific GL rendering context corresponding to the given GLContext

- **HWND fl_win32_xid (const FL_Window *win)**
  
  Returns the Windows-specific window reference corresponding to the given *FL_Window* object.

### 13.201.1 Detailed Description

Definitions of functions specific to the Windows platform.

### 13.202 win32.H

Go to the documentation of this file.
13.203 x.H

00001 // IMPORTANT: This file is deprecated since FLTK 1.4.0. DO NOT include it.
00002 // Fl/x.H will be removed in a future FLTK release.
00003 // Please include <FL/platform.H> instead if you really need it. See
00004 // documentation in FL/platform.H to decide whether you need that file.
00005
00006 #if !defined(Fl_X_H) && !defined(FL_DOXYGEN)
00007 # define Fl_X_H
00008 # include <FL/platform.H>
00009 #endif // !Fl_X_H

13.204 x11.H File Reference

Definitions of functions specific to the X11 platform.

Functions

- cairo_t * fl_cairo_gc ()
  Returns the Cairo-specific currently active graphics context (OPTION_USE_CAIRO=On)

- Display * fl_x11_display ()
  Returns the X11 Display in use.

- Fl_Window * fl_x11_find (Window xid)
  Returns the Fl_Window corresponding to the given Window reference.
• GC fl_x11_gc ()
  Returns the X11-specific currently active graphics context.
• void fl_x11_use_display (Display *d)
  Have FLTK use a pre-established X11 connection.
• Window fl_x11_xid (const Fl_Window *win)
  Returns the Window reference for the given Fl_Window, or zero if not shown().

13.204.1 Detailed Description
Definitions of functions specific to the X11 platform.

13.204.2 Function Documentation

13.204.2.1 fl_x11_find()

FL_Window * fl_x11_find (Window xid) [extern]
Returns the Fl_Window corresponding to the given Window reference.

13.204.2.2 fl_x11_gc()

GC fl_x11_gc () [extern]
Returns the X11-specific currently active graphics context.

13.204.2.3 fl_x11_use_display()

void fl_x11_use_display (Display *d) [extern]
Have FLTK use a pre-established X11 connection.
This function must be called before FLTK attempts to open its own X11 connection, that is, as long as fl_x11_display() returns NULL.

Parameters

```
d  the X11 Display* value representing a valid, pre-established X11 connection
```

13.204.2.4 fl_x11_xid()

Window fl_x11_xid (const Fl_Window * win) [extern]
Returns the Window reference for the given Fl_Window, or zero if not shown().

13.205 x11.H

Go to the documentation of this file.
#ifdef FL_DOXYGEN

extern Display *fl_x11_display();
extern void fl_x11_use_display(Display *d);
extern Window fl_x11_xid(const Fl_Window *win);
extern FL_EXPORT Window fl_x11_find(Window xid);
extern GC fl_x11_gc();
extern cairo_t* fl_cairo_gc();
#else // ! FL_DOXYGEN

#ifndef FL_PLATFORM_H
#error "Never use <FL/x11.H> directly; include <FL/platform.H> instead."
#endif // !FL_PLATFORM_H
#endif

#ifndef FL_PLATFORM_H
#error "Never use <FL/x11.H> directly; include <FL/platform.H> instead."
#endif // !FL_PLATFORM_H

typedef struct __GLXcontextRec *GLXContext;
extern GLXContext fl_x11_glcontext(GLContext rc);

const int info about the X server connection:
extern FL_EXPORT Display *fl_display;
extern FL_EXPORT Display *fl_x11_display();
extern FL_EXTERNAL void fl_x11_use_display(Display *);
extern FL_EXPORT Window fl_x11_xid(const Fl_Window *win);
extern FL_EXPORT Fl_Window *fl_x11_find(Window xid);
extern int fl_screen;
extern XVisualInfo *fl_visual;
extern Colormap fl_colormap;

// drawing functions:
extern FL_EXPORT GC fl_gc;
#ifdef FLTK_USE_CAIRO
typedef struct _cairo cairo_t;
extern FL_EXPORT cairo_t * fl_cairo_gc();
#endif
extern FL_EXPORT GC fl_x11_gc();
extern ulong fl_xpixel(Fl_Color i);
extern ulong fl_xpixel(uchar r, uchar g, uchar b);

// feed events into fltk:
extern FL_EXPORT int fl_handle(const XEvent&);
extern FL_EXPORT const XEvent* fl_xevent;
extern ulong fl_event_time;

#ifdef FL_LIBRARY || defined(FL_INTERNALS)
extern FL_EXPORT Window fl_message_window;
extern void *fl_xftfont;
#endif

// access to core fonts:
class FL_EXPORT Fl_XFont_On_Demand
{
public:

FL_XFont_On_Demand(Fl_Font * pp = NULL) : ptr(pp) { }
FL_XFont_On_Demand& operator=(const Fl_XFont_On_Demand& x)
{ ptr = x.ptr; return *this; }
FL_XFont_On_Demand& operator=(Fl_Font * p)
{ ptr = p; return *this; }
}

Generated by Doxygen
XFontStruct* value();
operator XFontStruct*() { return value(); }
operator XFontStruct& operator*() { return *value(); }
operator XFontStruct* operator->() { return value(); }
bool operator==(const Fl_XFont_On_Demand& x) { return ptr == x.ptr; }
bool operator!=(const Fl_XFont_On_Demand& x) { return ptr != x.ptr; }
private:
XFontStruct *ptr;
}

extern FL_EXPORT Fl_XFont_On_Demand fl_xfont;

extern FL_EXPORT char fl_override_redirect; // hack into Fl_X::make_xid()

#endif // FL_LIBRARY || FL_INTERNALS
#endif // FL_DOXYGEN

13.206 cgdebug.h

// OS X Core Graphics debugging help for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2010 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php
// This file allows easier debugging of Mac OS X Core Graphics
// but since it has proven to be tremendously useful in debugging
// I decided to add this file in case
// more bugs show up.
// This header is activated by adding the following
// line to "config.h"
// #include "src/cgdebug.h"
// Running "./configure" will remove this line from "config.h".
// When used erroneously, Core Graphics prints warnings to
// stderr. This is helpful, however it is not possible to
// associate a line number or source file with the warning message.
// This header file outputs a trace of CG calls, interweaving
// them with CG warnings.
// Matthias
#ifndef CGDEBUG
#define CGDEBUG
#include <stdio.h>
#include <Carbon/Carbon.h>
"+BitmapContextCreate
"+BitmapContextGetData
"+ClipCGContextToRegion
"+QDBeginCGContext
"+QDEndCGContext
/*+AddArc
/*+AddLineToPoint
/*+ClipToRect
/*+ClosePath
/*+ConcatCTM
/*+DrawImage
/*+FillPath
/*+FillRect
/*+Flush
/*+GetCTM
/*+MoveToPoint
/*+Release
/*+RestoreGState
/*+SaveGState
/*+ScaleCTM
/*+SetLineCap
/*+SetLineWidth
/*+SetLineJoin
Generated by Doxygen
inline OSStatus dbgLocation(const char *file, int line)
{
    fprintf(stderr, "%s:%d ", file, line);
    return 0;
}

inline OSStatus dbgEndl()
{
    fprintf(stderr, "\n");
    return 0;
}

inline void dbgCGContextClipToRect(CGContextRef a, CGRect b)
{
    CGContextClipToRect(a, b);
}

#define CGContextClipToRect(a, b) {
    fprintf(stderr, "%s:%d ", __FILE__, __LINE__); 
    dbgCGContextClipToRect(a, b);
    fprintf(stderr, "\n"); }

inline void dbgCGContextFillRect(CGContextRef a, CGRect b)
{
    CGContextFillRect(a, b);
}

#define CGContextFillRect(a, b) {
    fprintf(stderr, "%s:%d ", __FILE__, __LINE__); 
    dbgCGContextFillRect(a, b);
    fprintf(stderr, "\n"); }

inline OSStatus dbgQDEndCGContext(CGrafPtr a, CGContextRef *b)
{
    return QDEndCGContext(a, b);
}

#define QDEndCGContext(a, b) ( 
    dbgLocation(__FILE__, __LINE__) + 
    dbgQDEndCGContext(a, b) + 
    dbgEndl() )

inline OSStatus dbgQDBeginCGContext(CGrafPtr a, CGContextRef *b)
{
    return QDBeginCGContext(a, b);
}

#define QDBeginCGContext(a, b) ( 
   dbgLocation(__FILE__, __LINE__) + 
    dbgQDBeginCGContext(a, b) + 
    dbgEndl() )

inline void dbgClipCGContextToRegion(CGContextRef a, const Rect *b, RgnHandle c)
{
    ClipCGContextToRegion(a, b, c);
}

#define ClipCGContextToRegion(a, b, c) ( 
    fprintf(stderr, "%s:%d ", __FILE__, __LINE__); 
    dbgClipCGContextToRegion(a, b, c);
    fprintf(stderr, "\n"); }

inline void dbgCGContextMoveToPoint(CGContextRef context, float x, float y)
{
    CGContextMoveToPoint(context, x, y);
}

#define CGContextMoveToPoint(a, b, c) ( 
    fprintf(stderr, "%s:%d ", __FILE__, __LINE__); 
    dbgCGContextMoveToPoint(a, b, c);
    fprintf(stderr, "\n"); )

inline void dbgCGContextFillPath(CGContextRef context)
{
    CGContextFillPath(context);
}

#define CGContextFillPath(a) ( 

00154  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
00155  dbgCGContextClosePath(a); \
00156  fprintf(stderr, "\n"); } 
00157
00158 inline void dbgCGContextClosePath(CGContextRef context) 
00159 { 
00160  CGContextClosePath(context); 
00161 } 
00162
00163 #define CGContextClosePath(a) ( \n00164  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \n00165  dbgCGContextClosePath(a); \n00166  fprintf(stderr, "\n"); ) 
00167
00168 inline void dbgCGContextFlush(CGContextRef context) 
00169 { 
00170  CGContextFlush(context); 
00171 } 
00172
00173 #define CGContextFlush(a) ( \n00174  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \n00175  dbgCGContextFlush(a); \n00176  fprintf(stderr, "\n"); ) 
00177
00178 inline void dbgCGContextSaveGState(CGContextRef context) 
00179 { 
00180  CGContextSaveGState(context); 
00181 } 
00182
00183 #define CGContextSaveGState(a) ( \n00184  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \n00185  dbgCGContextSaveGState(a); \n00186  fprintf(stderr, "\n"); ) 
00187
00188 inline void dbgCGContextRestoreGState(CGContextRef context) 
00189 { 
00190  CGContextRestoreGState(context); 
00191 } 
00192
00193 #define CGContextRestoreGState(a) ( \n00194  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \n00195  dbgCGContextRestoreGState(a); \n00196  fprintf(stderr, "\n"); ) 
00197
00198 
00199 #endif 
00200

13.207 fastarrow.h

00001 #define fastarrow_width 16
00002 #define fastarrow_height 16
00003 static const unsigned char fastarrow_bits[] = {
00004 0xe0, 0xe0, 0x00, 0x07, 0xe0, 0xe0, 0x07, 0xe0, 0x3f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
00005 0xe0, 0xe0, 0x00, 0x07, 0xe0, 0xe0, 0x07, 0xe0, 0x3f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
00006 0xe0, 0x3f, 0xe0, 0x07, 0xe0, 0xe0, 0x00, 0x00, 0x00, 0x00};

13.208 Fl.cxx File Reference

Implementation of the member functions of class Fl.
#include <FL/Fl.H>
#include <FL/platform.H>
#include "FL_Screen_Driver.H"
#include "FL_Window_Driver.H"
#include "FL_System_Driver.H"
#include "FL_Timeout.h"
#include <FL/Fl_Window.H>
#include <FL/Fl_Tooltip.H>
#include <FL/fl_draw.H>
#include <ctype.h>
#include <stdlib.h>
#include "flstring.h"
Macros

- \#define FOREVER 1e20

Functions

- bool fl_clipboard_notify_empty (void)
- void fl_close_display ()
- const char * fl_filename_name (const char *name)
  
  Gets the file name from a path.
- FL_Window * fl_find (Window xid)

  Returns the FL_Window that corresponds to the given window reference, or NULL if not found.
- void fl_fix_focus ()
- void fl_open_callback (void(*)(const char *cb))(const char *)
  
  Register a function called for each file dropped onto an application icon.
- void fl_open_display ()

  Opens the display.
- int fl_send_system_handlers (void *e)
- void fl_throw_focus (Fl_Widget *w)
- void fl_trigger_clipboard_notify (int source)
- Window fl_xid_(const Fl_Window *w)

Variables

- bool fl_disable_wayland = true
  
  Prevent the FLTK library from using its Wayland backend and forces it to use its X11 backend.
- const char * fl_local_alt = Fl::system_driver()->alt_name()
  
  string pointer used in shortcuts, you can change it to another language
- const char * fl_local_ctrl = Fl::system_driver()->control_name()
  
  string pointer used in shortcuts, you can change it to another language
- int(* fl_local_grab )(int)
- const char * fl_local_meta = Fl::system_driver()->meta_name()
  
  string pointer used in shortcuts, you can change it to another language
- const char * fl_local_shift = Fl::system_driver()->shift_name()
  
  string pointer used in shortcuts, you can change it to another language
- Fl_Widget * fl_oldfocus
- Fl_Widget * fl_selection_requestor

13.208.1 Detailed Description

Implementation of the member functions of class Fl.

13.208.2 Function Documentation

13.208.2.1 fl_find()

FL_Window * fl_find (Window xid)

Returns the FL_Window that corresponds to the given window reference, or NULL if not found.

Deprecated  

Kept in the X11, Windows, and macOS platforms for compatibility with FLTK versions before 1.4. Please use fl_x11_find(Window), fl_wl_find(struct wid_window*), fl_win32_find(HWND) or fl_mac_find(FLWindow*) with FLTK 1.4.0 and above.
13.208.2.2 \texttt{fl\_open\_display()}

\begin{verbatim}
void fl_open_display()
Opens the display.
Automatically called by the library when the first window is show()’n. Does nothing if the display is already open.
\end{verbatim}

13.208.3 Variable Documentation

13.208.3.1 \texttt{fl\_disable\_wayland}

\begin{verbatim}
bool fl_disable_wayland = true
Prevent the FLTK library from using its Wayland backend and forces it to use its X11 backend.
Put this declaration somewhere in your code outside the body of any function :
FL_EXPORT bool fl_disable_wayland = true;
This declaration makes sure source code developed for FLTK 1.3, including X11-specific code, will build and run
with FLTK 1.4 and its Wayland platform with this single source code level change. This declaration has no effect on
non-Wayland platforms. Don’t put this declaration if you want the Wayland backend to be used when it’s available.
\end{verbatim}

13.209 \texttt{fl\_arc.cxx} File Reference

Utility functions for drawing arcs and circles.
\begin{verbatim}
#include <FL/fl_draw.H>
#include <FL/math.h>
\end{verbatim}

13.209.1 Detailed Description

Utility functions for drawing arcs and circles.

13.210 \texttt{fl\_ask.cxx} File Reference

Utility functions for common dialogs.
\begin{verbatim}
#include <FL/Fl.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Input_.H>
#include "flstring.h"
#include "Fl_Screen_Driver.H"
#include <FL/fl_ask.H>
#include "Fl_Message.h"
#include <stdio.h>
#include <stdarg.h>
\end{verbatim}

\textbf{Functions}

- void \texttt{fl\_alert} (const char *fmt, ...)
  \emph{Shows an alert message dialog box.}

- int \texttt{fl\_ask} (const char *fmt, ...)
  \emph{Shows a dialog displaying the fmt message, this dialog features 2 yes/no buttons.}

- void \texttt{fl\_beep} (int type)
  \emph{Emits a system beep.}

- int \texttt{fl\_choice} (const char *fmt, const char *b0, const char *b1, const char *b2, ...)
  \emph{Shows a dialog displaying the printf style fmt message.}

- int \texttt{fl\_choice\_n} (const char *fmt, const char *b0, const char *b1, const char *b2, ...)
  \emph{Shows a dialog displaying the printf style fmt message.}

- const char * \texttt{fl\_input} (const char *fmt, const char *defstr, ...)

Generated by Doxygen
Shows an input dialog displaying the `fmt` message with variable arguments.

- `const char * fl_input (int maxchar, const char *fmt, const char *defstr,...)`
  Shows an input dialog displaying the `fmt` message with variable arguments.

- `void fl_message (const char *fmt,....)`
  Shows an information message dialog box.

- `int fl_message_hotspot ()`
  Gets whether or not to move the message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()` to follow the mouse pointer.

- `void fl_message_hotspot (int enable)`
  Sets whether or not to move the message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()` to follow the mouse pointer.

- `Fl_Widget * fl_message_icon ()`
  Gets the Fl_Box icon container of the current default dialog used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`.

- `void fl_message_icon_label (const char *str)`
  Sets the icon label of the dialog window used in many common dialogs.

- `void fl_message_position (const int x, const int y, const int center)`
  Sets the preferred position for the message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`.

- `void fl_message_position (Fl_Widget *widget)`
  Sets the preferred position for the message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`.

- `int fl_message_position (int *x, int *y)`
  Gets the preferred position for the message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`.

- `void fl_message_title (const char *title)`
  Sets the title of the dialog window used in many common dialogs.

- `void fl_message_title_default (const char *title)`
  Sets the default title of the dialog window used in many common dialogs.

- `const char * fl_password (const char *fmt, const char *defstr,...)`
  Shows an input dialog displaying the `fmt` message with variable arguments.

- `const char * fl_password (int maxchar, const char *fmt, const char *defstr,...)`
  Shows an input dialog displaying the `fmt` message with variable arguments.

Variables

- `const char * fl_cancel = "Cancel"`
  string pointer used in common dialogs, you can change it to another language

- `const char * fl_close = "Close"`
  string pointer used in common dialogs, you can change it to another language

- `Fl_Font fl_message_font_ = FL_HELVETICA`
- `Fl_Fontsize fl_message_size_ = -1`
- `const char * fl_no = "No"`
  string pointer used in common dialogs, you can change it to another language

- `const char * fl_ok = "OK"`
  string pointer used in common dialogs, you can change it to another language

- `const char * fl_yes = "Yes"`
  string pointer used in common dialogs, you can change it to another language
13.210.1 Detailed Description

Utility functions for common dialogs.
This file defines the functions

- `fl_alert()
- `fl_beep()
- `fl_message()
- `fl_ask()
- `fl_choice()
- `fl_input()
- `fl_input_str()
- `fl_password()
- `fl_password_str()

and some more functions to change their behavior (positioning, window title, and more).
Since FLTK 1.4.0 a big part of these functions is implemented in class Fl_Message.

13.211 fl_boxtype.cxx File Reference

Drawing code for common box types.
#include <FL/Fl.H>
#include <FL/Fl_Widget.H>
#include <FL/fl_draw.H>
#include <config.h>

Macros

- `#define D1 BORDER_WIDTH
- `#define D2 (BORDER_WIDTH+BORDER_WIDTH)
- `#define fl_border_box fl_rectbound
  allow consistent naming

Functions

- `void fl_border_frame (int x, int y, int w, int h, Fl_Color c)
  Draws a frame of type FL_BORDER_FRAME.
- `void fl_down_box (int x, int y, int w, int h, Fl_Color c)
  Draws a box of type FL_DOWN_BOX.
- `void fl_down_frame (int x, int y, int w, int h, Fl_Color)
  Draws a frame of type FL_DOWN_FRAME.
- `void fl_draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c)
  Draws a box using given type, position, size and color.
- `void fl_embossed_box (int x, int y, int w, int h, Fl_Color c)
  Draws a box of type FL_EMBOSSED_BOX.
- `void fl_embossed_frame (int x, int y, int w, int h, Fl_Color)
  Draws a frame of type FL_EMBOSSED_FRAME.
- `void fl_engraved_box (int x, int y, int w, int h, Fl_Color c)
  Draws a box of type FL_ENGRAVED_BOX.
- `void fl_engraved_frame (int x, int y, int w, int h, Fl_Color)
Draws a frame of type FL_ENGRAVED_FRAME.

- **void fl_flat_box** (int x, int y, int w, int h, Fl_Color c)
  
  Draws a box of type FL_FLAT_BOX.

- **void fl_frame** (const char *s, int x, int y, int w, int h)
  
  Draws a series of line segments around the given box.

- **void fl_frame2** (const char *s, int x, int y, int w, int h)
  
  Draws a series of line segments around the given box.

- **const uchar * fl_gray_ramp ()**

- **void fl_internal_boxtype (Fl_Boxtype t, Fl_Box_Draw_F *f)**
  
  Sets the drawing function for a given box type.

- **void fl_no_box** (int, int, int, int, Fl_Color)
  
  Draws a box of type FL_NO_BOX.

- **void fl_rectbound** (int x, int y, int w, int h, Fl_Color bgcolor)
  
  Draws a bounded rectangle with a given position, size and color.

- **void fl_thin_down_box** (int x, int y, int w, int h, Fl_Color c)
  
  Draws a box of type FL_THIN_DOWN_BOX.

- **void fl_thin_down_frame** (int x, int y, int w, int h, Fl_Color)
  
  Draws a frame of type FL_THIN_DOWN_FRAME.

- **void fl_thin_up_box** (int x, int y, int w, int h, Fl_Color c)
  
  Draws a box of type FL_THIN_UP_BOX.

- **void fl_thin_up_frame** (int x, int y, int w, int h, Fl_Color)
  
  Draws a frame of type FL_UP_BOX.

- **void fl_up_box** (int x, int y, int w, int h, Fl_Color c)
  
  Draws a box of type FL_UP_BOX.

- **void fl_up_frame** (int x, int y, int w, int h, Fl_Color)
  
  Draws a frame of type FL_UP_FRAME.

### 13.211.1 Detailed Description

Drawing code for common box types.

### 13.211.2 Function Documentation

#### 13.211.2.1 fl_internal_boxtype()

```c
void fl_internal_boxtype ( 
    Fl_Boxtype t, 
    Fl_Box_Draw_F * f )
```

Sets the drawing function for a given box type.

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>t</td>
<td>box type</td>
</tr>
<tr>
<td>in</td>
<td>f</td>
<td>box drawing function</td>
</tr>
</tbody>
</table>

#### 13.211.2.2 fl_rectbound()

```c
void fl_rectbound ( 
    int x, 
    int y, 
    int w, 
    int h, 
    Fl_Color bgcolor )
```
Draws a bounded rectangle with a given position, size and color.
Equivalent to drawing a box of type FL_BORDER_BOX.

13.212  fl_cmap.h

00001 //
00002 // DO NOT EDIT THIS FILE!
00003 //
00004 // This file must be generated by "util/cmap.cxx".
00005 // See instructions in this file.
00006 //
00007 // Copyright 1998-2022 by Bill Spitzak and others.
00008 //
00009 // This library is free software. Distribution and use rights are outlined in
00100 // the file "COPYING" which should have been included with this file. If this
00101 // file is missing or damaged, see the license at:
00102 //
00103 // https://www.fltk.org/COPYING.php
00104 //
00105 // Please see the following page on how to report bugs and issues:
00106 //
00107 // https://www.fltk.org/bugs.php
00108 //
00109 0x00000000, // 0
00110 0xff000000, // 1
00111 0x00ff0000, // 2
00112 0xffff0000, // 3
00113 0x0000ff00, // 4
00114 0xff00ff00, // 5
00115 0x00ffff00, // 6
00116 0xffffff00, // 7
00117 0x55555500, // 8
00118 0xc6717100, // 9
00119 0x71c67100, // 10
00120 0x8e8e3800, // 11
00121 0x7171c600, // 12
00122 0x8e388e00, // 13
00123 0x8e8e8e00, // 14
00124 0x00000000, // 15
00125 0x00ff0000, // 16
00126 0x0000ff00, // 17
00127 0x00ffff00, // 18
00128 0x00000000, // 19
00129 0x00ff0000, // 20
00130 0x0000ff00, // 21
00131 0x00ffff00, // 22
00132 0x00000000, // 23
00133 0x00ff0000, // 24
00134 0x0000ff00, // 25
00135 0x00ffff00, // 26
00136 0x00000000, // 27
00137 0x00ff0000, // 28
00138 0x0000ff00, // 29
00139 0x00ffff00, // 30
00140 0x00000000, // 31
00141 0x00ff0000, // 32
00142 0x0000ff00, // 33
00143 0x00ffff00, // 34
00144 0x00000000, // 35
00145 0x00ff0000, // 36
00146 0x0000ff00, // 37
00147 0x00ffff00, // 38
00148 0x00000000, // 39
00149 0x00ff0000, // 40
00150 0x0000ff00, // 41
00151 0x00ffff00, // 42
00152 0x00000000, // 43
00153 0x00ff0000, // 44
00154 0x0000ff00, // 45
00155 0x00ffff00, // 46
00156 0x00000000, // 47
00157 0x00ff0000, // 48
00158 0x0000ff00, // 49
00159 0x00ffff00, // 50
00160 0x00000000, // 51
00161 0x00ff0000, // 52
00162 0x0000ff00, // 53
00163 0x00ffff00, // 54
00164 0x00000000, // 55
00165 0x00ff0000, // 56
00166 0x0000ff00, // 57
00167 0x00ffff00, // 58
00168 0x00000000, // 59
00169 0x00ff0000, // 60
00080 0x00b60000, // 61
00081 0x00da0000, // 62
00082 0x00ff0000, // 63
00083 0x3f000000, // 64
00084 0x3f240000, // 65
00085 0x3f480000, // 66
00086 0x3f6d0000, // 67
00087 0x3f910000, // 68
00088 0x3fb60000, // 69
00089 0x3fda0000, // 70
00090 0x3fff0000, // 71
00091 0x7f000000, // 72
00092 0x7f240000, // 73
00093 0x7f480000, // 74
00094 0x7f6d0000, // 75
00095 0x7f910000, // 76
00096 0x7fb60000, // 77
00097 0x7fda0000, // 78
00098 0x7fff0000, // 79
00099 0xb0000000, // 80
00100 0xb0240000, // 81
00101 0xb0480000, // 82
00102 0xb06d0000, // 83
00103 0xb0910000, // 84
00104 0xb0bf0000, // 85
00105 0xb0fda0000, // 86
00106 0xb0ff0000, // 87
00107 0xbff00000, // 88
00108 0xbf240000, // 89
00109 0xbf480000, // 90
00110 0xbf6d0000, // 91
00111 0xbf910000, // 92
00112 0xbfb60000, // 93
00113 0xbff00000, // 94
00114 0xbff00000, // 95
00115 0xb0003f00, // 96
00116 0xb0243f00, // 97
00117 0xb0483f00, // 98
00118 0xb06d3f00, // 99
00119 0xb0913f00, // 100
00120 0xb0bf3f00, // 101
00121 0xb0fda3f00, // 102
00122 0xb0ff3f00, // 103
00123 0xbff03f00, // 104
00124 0xbf243f00, // 105
00125 0xbf483f00, // 106
00126 0xbf6d3f00, // 107
00127 0xbf913f00, // 108
00128 0xbfb63f00, // 109
00129 0xbff03f00, // 110
00130 0xbff03f00, // 111
00131 0xb0003f00, // 112
00132 0xb0243f00, // 113
00133 0xb0483f00, // 114
00134 0xb06d3f00, // 115
00135 0xb0913f00, // 116
00136 0xb0bf3f00, // 117
00137 0xb0fda3f00, // 118
00138 0xb0ff3f00, // 119
00139 0xbff03f00, // 120
00140 0xbff03f00, // 121
00141 0xbff03f00, // 122
00142 0xb06d3f00, // 123
00143 0xb0fda3f00, // 124
00144 0xb0ff3f00, // 125
00145 0xbff03f00, // 126
00146 0xbff03f00, // 127
00147 0xbff03f00, // 128
00148 0xbff03f00, // 129
00149 0xbff03f00, // 130
00150 0xbff03f00, // 131
00151 0xbff03f00, // 132
00152 0xbff03f00, // 133
00153 0xbff03f00, // 134
00154 0xbff03f00, // 135
00155 0xb0007f00, // 136
00156 0xb0247f00, // 137
00157 0xb0487f00, // 138
00158 0xb06d7f00, // 139
00159 0xb0917f00, // 140
00160 0xb0b67f00, // 141
00161 0xb0fa7f00, // 142
00162 0xb0ff7f00, // 143
00163 0xb3007f00, // 144
00164 0xb3247f00, // 145
00165 0xb3487f00, // 146
00166 0xb36d7f00, // 147
13.213 fl_color.cxx File Reference

Color handling.
#include <FL/Fl.H>
#include <FL/Fl_Device.H>
#include <FL/Fl_Graphics_Driver.H>
#include "fl_cmap.h"

Functions

- Fl_Color fl_color_average (Fl_Color color1, Fl_Color color2, float weight)
  Returns the weighted average color between the two given colors.
- Fl_Color fl_inactive (Fl_Color c)
  Returns the inactive, dimmed version of the given color.

Variables

- unsigned fl_cmap [256]

13.213.1 Detailed Description

Color handling.

13.213.2 Variable Documentation

13.213.2.1 fl_cmap

unsigned f1_cmap[256]
Initial value:
= {
}

13.214 FlCompose.cxx File Reference

Utility functions to support text input.
#include <FL/Fl.H>
#include "Fl_Screen_Driver.H"
13.214.1 Detailed Description
Utility functions to support text input.

13.215 fl_contrast.cxx File Reference
Color contrast handling.
#include <FL/Fl.H>
#include <math.h>

Functions
- Fl_Color fl_contrast (Fl_Color fg, Fl_Color bg, Fl_Fontsize fs, int context)
  Returns a color that contrasts with the background color.
- void fl_contrast_function (Fl_Contrast_Function *f)
  Register a custom contrast function.
- int fl_contrast_level ()
  Get the contrast level (sensitivity) of the fl_contrast() method.
- void fl_contrast_level (int level)
  Set the contrast level (sensitivity) of the fl_contrast() method.
- int fl_contrast_mode ()
  Return the current contrast algorithm (mode).
- void fl_contrast_mode (int mode)
  Set the contrast algorithm (mode).
- double fl_lightness (Fl_Color color)
  Return the perceived lightness of a color.
- double fl_luminance (Fl_Color color)
  Return the raw / physical luminance of a color.
- unsigned get_color (Fl_Color i)

Variables
- unsigned fl_cmap [256]

13.215.1 Detailed Description
Color contrast handling.
Implementation of fl_contrast() and its variants.

13.216 fl_curve.cxx File Reference
Utility for drawing Bézier curves, adding the points to the current fl_begin/fl_vertex/fl_end path.
#include <FL/fl_draw.H>
#include <math.h>

13.216.1 Detailed Description
Utility for drawing Bézier curves, adding the points to the current fl_begin/fl_vertex/fl_end path.
Incremental math implementation: I very much doubt this is optimal! From Foley/vanDam page 511. If anybody has a better algorithm, please send it!
13.217  **Fl_Double_Window.cxx File Reference**

**Fl_Double_Window** implementation.

```c
#include <FL/Fl.H>
#include <FL/platform.H>
#include <FL/Fl_Double_Window.H>
#include <FL/fl_draw.H>
#include "Fl_Window_Driver.H"
```

13.217.1  Detailed Description

**Fl_Double_Window** implementation.

13.218  **Fl_get_system_colors.cxx File Reference**

System color support.

```c
#include <FL/Fl.H>
#include "Fl_Screen_Driver.H"
#include "Fl_System_Driver.H"
#include <FL/fl_draw.H>
#include <FL/platform.H>
#include <FL/math.h>
#include <FL/fl_utf8.h>
#include <FL/fl_string_functions.h>
#include "flstring.h"
#include <stdio.h>
#include <stdlib.h>
#include <FL/Fl_Pixmap.H>
#include <FL/Fl_Tiled_Image.H>
#include "tile.xpm"
```

**Macros**

- `#define D1 BORDER_WIDTH`
- `#define D2 (BORDER_WIDTH+BORDER_WIDTH)`

**Functions**

- `void fl_down_box (int, int, int, int, Fl_Color)`
  
  Draws a box of type FL_DOWN_BOX.

- `void fl_down_frame (int, int, int, int, Fl_Color)`
  
  Draws a frame of type FL_DOWN_FRAME.

- `int fl_parse_color (const char *p, uchar &r, uchar &g, uchar &b)`

  Parses a string containing a description of a color and write r, g, and b.

- `void fl_round_down_box (int, int, int, int, Fl_Color)`

- `void fl_round_up_box (int, int, int, int, Fl_Color)`

- `void fl_thin_down_box (int, int, int, int, Fl_Color)`

  Draws a box of type FL_THIN_DOWN_BOX.

- `void fl_thin_down_frame (int, int, int, int, Fl_Color)`

  Draws a frame of type FL_THIN_DOWN_FRAME.

- `void fl_thin_up_box (int, int, int, int, Fl_Color)`

  Draws a box of type FL_THIN_UP_BOX.

- `void fl_thin_up_frame (int, int, int, int, Fl_Color)`

  Draws a frame of type FL_THIN_UP_FRAME.
• **void fl_up_box** (int, int, int, int, Fl_Color)
  Draws a box of type FL_UP_BOX.

• **void fl_up_frame** (int, int, int, int, Fl_Color)
  Draws a frame of type FL_UP_FRAME.

### Variables

• const char * fl_bg = NULL
• const char * fl_bg2 = NULL
• const char * fl_fg = NULL

### 13.218.1 Detailed Description

System color support.

### 13.218.2 Function Documentation

#### 13.218.2.1 fl_parse_color()

```c
int fl_parse_color (const char * p, uchar & r, uchar & g, uchar & b )
```

Parse a string containing a description of a color and write r, g, and b.

This call is used by the Pixmap file format interpreter and by the command line arguments parser to set UI colors. RGB color triplets usually start with a '#' character, but it can be omitted if it does not conflict with the later rules. Color components are defined in hexadecimal notation with 1, 2, 3, or four hex digits per component, making color triplets 3, 6, 9, or 12 characters long. The interpreter is case insensitive. Valid code examples include "FF0000" for red, "#0F0" for green, and "000000004444" for a dark blue.

On the X11 platform, color values can also be given a color name like "red". The list of accepted color names is provided by the X11 server.

If none of the color interpretations work, `fl_parse_color` returns 0. The Pixmap reader interprets those as transparent, and are usually written as "None", "#transparent", or "bg".

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>p</th>
<th>a C-string describing the color</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>r,g,b</td>
<td>the color components in the 0...255 range</td>
</tr>
</tbody>
</table>

#### Returns

0 if the color cannot be interpreted, 1 otherwise

### 13.219 FL_Gl_Choice.H

00001 //
00002 // OpenGL definitions for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2018 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file COPYING which should have been included with this file. If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //    https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //    https://www.fltk.org/bugs.php
00015 //
00016 //
00017 // Internal interface to set up OpenGL.
A "Fl_Gl_Choice" is created from an OpenGL mode and holds information necessary to create a window (on X) and to create an OpenGL "context" (on both X and Win32).

create_gl_context takes a window (necessary only on Win32) and an Fl_Gl_Choice and returns a new OpenGL context. All contexts share display lists with each other.

On X another create_gl_context is provided to create it for any X visual.

create_gl_context takes a window (necessary only on Win32) and an Fl_Gl_Choice and returns a new OpenGL context. All contexts share display lists with each other.

don't do this themselves...

delete_gl_context destroys the context.

This code is used by Fl_Gl_Window, gl_start(), and gl_visual()

This code is used by Fl_Gl_Window, gl_start(), and gl_visual()
void alist(const int *l) { pWindow->alist = l;}
void overlay() { return pWindow->overlay;}
void draw_overlay() { pWindow->draw_overlay();}

Fl_Gl_Window_Driver(Fl_Gl_Window *win) : pWindow(win) { current_prog=0; } 
virtual ~Fl_Gl_Window_Driver() {}
static Fl_Gl_Window_Driver *newGlWindowDriver(Fl_Gl_Window *w);
static Fl_Gl_Window_Driver *global();
virtual float pixels_per_unit() { return 1; }
virtual void before_show(int&) {}
virtual void after_show() {}
virtual void invalidate();
virtual int mode_(int /*m*/, const int * /*a*/) {return 0;}
virtual void make_current_before() {}
virtual void make_current_after() {}
virtual void swap_buffers() {}
virtual void resize(int /*is_a_resize*/, int /*w*/, int /*h*/) {}
virtual void swap_type();
virtual void swap_interval(int) { }
virtual int swap_interval() const { return -1; }
virtual Fl_Gl_Choice *find(int /*mode*/, const int * /*alistp*/) {return NULL;}
virtual GLContext create_gl_context(Fl_Window *, const Fl_Gl_Choice*) {return 0;}
virtual void set_gl_context(Fl_Window *, GLContext) {}
virtual void delete_gl_context(GLContext) {}
virtual void make_overlay(void * &o);
virtual void hide_overlay() {} // the default implementation may be enough
virtual void make_overlay_current() {}
virtual void redraw_overlay() {}
virtual int can_do_overlay() {return 0;}
virtual int do_overlay() { return 0;}
virtual void gl_hide_before(void *&) {} // the default implementation may be enough
virtual void gl_visual(Fl_Gl_Choice *); // support for Fl::gl_visual() function
virtual void gl_start() {} // support for gl_start() function
virtual void draw_string_legacy(const char * str, int n); // support for Fl::gl_draw() without textures
virtual int genlistsize() { return 0; } // support for Fl::gl_draw()
virtual Fl_Font_Descriptor ** fontnum_to_fontdescriptor(int fnum);
13.222.1 Detailed Description
Implementation of class Fl_Graphics_Driver.

13.222 Fl_Grid.cxx File Reference

Implements the Fl_Grid container widget.
#include <FL/Fl_Grid.H>
#include <FL/fl_draw.H>

13.222.1 Detailed Description
Implements the Fl_Grid container widget.

Since
1.4.0

13.223 Fl_Image_Reader.h

00001 //
00002 // Internal (Image) Reader class for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 2020-2021 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file. If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 // https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 // https://www.fltk.org/bugs.php
00015 //
00016 //
00017 /*
00018 This internal (undocumented) class reads data chunks from a file or from
00019 memory in LSB-first byte order.
00020 This class is used in Fl_GIF_Image and Fl_BMP_Image to avoid code
00021 duplication and may be extended to be used in similar cases. Future
00022 options might be to read data in MSB-first byte order or to add more
00023 methods.
00024 */
00025#
00026 ifndef FL_IMAGE_READER_H
00027 #define FL_IMAGE_READER_H
00028
00030 #include <stdio.h>
00031
00032 class Fl_Image_Reader {
00033 public:
00034 // Create the reader.
00035 Fl_Image_Reader();
00036 : is_file_(0),
00037 , is_data_(0),
00038 , file_(0L),
00039 , data_(0L),
00040 , start_(0L),
00041 , end_((const unsigned char *)(-1L)),
00042 , name_(0L),
00043 , error_(0) {};
00044
00045 // Initialize the reader to access the file system, filename is copied
00046 // and stored.
00047 int open(const char *filename);
00048
00049 // Initialize the reader for memory access, name is copied and stored
00050 int open(const char *imagename, const unsigned char *data, const size_t datasize);
00051 // Deprecated, DO NOT USE!
00052 int open(const char *imagename, const unsigned char *data);
00053
00054 // Close and destroy the reader
00055 ~Fl_Image_Reader();
00056

Generated by Doxygen
// Read a single byte from memory or a file
unsigned char read_byte();

// Read a 16-bit unsigned integer, LSB-first
unsigned short read_word();

// Read a 32-bit unsigned integer, LSB-first
unsigned int read_dword();

// Read a 32-bit signed integer, LSB-first
int read_long() { return (int)read_dword(); }

// Move the current read position to a byte offset from the beginning
// of the file or the original start address in memory
void seek(unsigned int n);

// Get the current file or memory offset from the beginning
// of the file or the original start address in memory
long tell() const;

// Get the current EOF or error status of the file or data block
int error() const { return error_; }

// return the name or filename for this reader
const char *name() const { return name_; }

// skip a given number of bytes
void skip(unsigned int n) { seek((unsigned int)tell() + n); }

private:
// open() sets this if we read from a file
char is_file_;
// open() sets this if we read from memory
char is_data_;
// a pointer to the opened file
FILE *file_;
// a pointer to the current byte in memory
const unsigned char *data_;
// a pointer to the start of the image data
const unsigned char *start_;
// a pointer to the end of image data if reading from memory, otherwise undefined
// note: currently (const unsigned char *)(-1L) if end of memory is not available
// ... which means "unlimited"
const unsigned char *end_;
// a flag to store EOF or error status
int error_;
const char *name_;

#include <FL/Fl_Export.H>

class FL_EXPORT Fl_Int_Vector {
  int *arr_;
  unsigned int size_;

  void init() {
    arr_ = 0;
    size_ = 0;
  }

  void copy(int *newarr, unsigned int newsize);
};

// An STL-ish vector without templates for the Fast Light Tool Kit (FLTK).
// Copyright 2002 by Greg Ercolano.
// Copyright 2022-2023 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

class Fl_Int_Vector {
  int *arr_;
  unsigned int size_;

  void init() {
    arr_ = 0;
    size_ = 0;
  }

  void copy(int *newarr, unsigned int newsize);
```cpp
public:
    Fl_Int_Vector() {
        init();
    }
    ~Fl_Int_Vector();
    Fl_Int_Vector(Fl_Int_Vector &o) {
        init();
        copy(o.arr_, o.size_);
    }
    Fl_Int_Vector &operator=(Fl_Int_Vector &o) {
        init();
        copy(o.arr_, o.size_);
        return *this;
    }
    int operator[](int x) const {
        return arr_[x];
    }
    int &operator[](int x) {
        return arr_[x];
    }
    unsigned int size() const {
        return size_;
    }
    void size(unsigned int count);
    int pop_back() {
        int tmp = arr_[size_ - 1];
        size_--;
        return tmp;
    }
    void push_back(int val) {
        unsigned int x = size_;
        size_(size_ + 1);
        arr_[x] = val;
    }
    int back() const {
        return arr_[size_ - 1];
    }
    bool empty() const {
        return (size_ == 0) ? true : false;
    }

#endif // Fl_Int_Vector_H
```

---

13.225 Fl_Message.h

```cpp
// Common dialog header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2022 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php
#ifndef _src_Fl_Message_h_
#define _src_Fl_Message_h_
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/fl_ask.H>

class Fl_Button;
class Fl_Input;

/* Note: Do not FL_EXPORT this class, it's for internal use only */
```
class Fl_Message_Box : public Fl_Box {
public:
  Fl_Message_Box(int X, int Y, int W, int H) : Fl_Box(X, Y, W, H) {}
  int handle(int e) FL_OVERRIDE;
}; // class Fl_Message_Box

class Fl_Message {
private:
  static Fl_Box *message_icon_; // returned by Fl_Message::message_icon()
  static const char *message_title_; // static variables and methods
  static const char *message_title_default_;
  static const char *message_icon_label_; // icon label for next dialog (STR #2763)
  static const char *message_icon_label_; // size of allocated text buffer
  static int input_size_; // points to the allocated text buffer
  static char *input_buffer_; // static void button_cb_(Fl_Widget *w, void *d);
  static void window_cb_(Fl_Widget *w, void *d);
public:
  static Fl_Box *message_icon();
  static void message_title(const char *title);
  static void message_title_default(const char *title);
  static void icon_label(const char *str);
  static void message_position(const int x, const int y, const int center) {
    form_x_ = x;
    form_y_ = y;
    form_position_ = center ? 2 : 1;
  }
  static void message_position(Fl_Widget *widget) {
    int xo, yo;
    Fl_Window *win = widget->top_window_offset(xo, yo);
    form_x_ = xo + widget->w() / 2;
    form_y_ = yo + widget->h() / 2;
    if (win) {
      form_x_ += win->x();
      form_y_ += win->y();
    }
    form_position_ = 2;
  }
  static int message_position(int *x, int *y) {
    *x = form_position_ ? form_x_ : -1;
    *y = form_position_ ? form_y_ : -1;
    return form_position_;
  }
  static void message_hotspot(int enable) { enable_hotspot_ = enable ? 1 : 0; }
  static int message_hotspot() { return enable_hotspot_; }
  int window_closed() const {
    return window_closed_;}
  static void message_position(Fl_Widget *widget) {
    int xo, yo;
    Fl_Window *win = widget->top_window_offset(xo, yo);
    form_x_ = xo + widget->w() / 2;
    form_y_ = yo + widget->h() / 2;
    if (win) {
      form_x_ += win->x();
      form_y_ += win->y();
    }
    form_position_ = 2;
  }
  static int message_position(int *x, int *y) {
    if (*x) {
      *x = form_position_ ? form_x_ : -1;
      *y = form_position_ ? form_y_ : -1;
      return form_position_;
    }
  }
  static void message_hotspot(int enable) { enable_hotspot_ = enable ? 1 : 0; }
  static int message_hotspot() { return enable_hotspot_; }
  int window_closed() const {
    return window_closed_;}
  static void message_position(Fl_Widget *widget) {
    int xo, yo;
    Fl_Window *win = widget->top_window_offset(xo, yo);
    form_x_ = xo + widget->w() / 2;
    form_y_ = yo + widget->h() / 2;
    if (win) {
      form_x_ += win->x();
      form_y_ += win->y();
    }
    form_position_ = 2;
  }
  static int message_position(int *x, int *y) {
    if (*x) {
      *x = form_position_ ? form_x_ : -1;
      *y = form_position_ ? form_y_ : -1;
      return form_position_;
    }
  }
  static void message_hotspot(int enable) { enable_hotspot_ = enable ? 1 : 0; }
  static int message_hotspot() { return enable_hotspot_; }
  int window_closed() const {
    return window_closed_;}
  static void message_position(Fl_Widget *widget) {
    int xo, yo;
    Fl_Window *win = widget->top_window_offset(xo, yo);
    form_x_ = xo + widget->w() / 2;
    form_y_ = yo + widget->h() / 2;
    if (win) {
      form_x_ += win->x();
      form_y_ += win->y();
    }
    form_position_ = 2;
  }
  static int message_position(int *x, int *y) {
    if (*x) {
      *x = form_position_ ? form_x_ : -1;
      *y = form_position_ ? form_y_ : -1;
      return form_position_;
    }
  }
  static void message_hotspot(int enable) { enable_hotspot_ = enable ? 1 : 0; }
  static int message_hotspot() { return enable_hotspot_; }
  int window_closed() const {
    return window_closed_;}
  static void message_position(Fl_Widget *widget) {
    int xo, yo;
    Fl_Window *win = widget->top_window_offset(xo, yo);
    form_x_ = xo + widget->w() / 2;
    form_y_ = yo + widget->h() / 2;
    if (win) {
      form_x_ += win->x();
      form_y_ += win->y();
    }
    form_position_ = 2;
  }
  static int message_position(int *x, int *y) {
    if (*x) {
      *x = form_position_ ? form_x_ : -1;
      *y = form_position_ ? form_y_ : -1;
      return form_position_;
    }
  }
  static void message_hotspot(int enable) { enable_hotspot_ = enable ? 1 : 0; }
  static int message_hotspot() { return enable_hotspot_; }
  int window_closed() const {
    return window_closed_;}
  static void message_position(Fl_Widget *widget) {
    int xo, yo;
    Fl_Window *win = widget->top_window_offset(xo, yo);
    form_x_ = xo + widget->w() / 2;
    form_y_ = yo + widget->h() / 2;
    if (win) {
      form_x_ += win->x();
      form_y_ += win->y();
    }
    form_position_ = 2;
  }
  static int message_position(int *x, int *y) {
    if (*x) {
      *x = form_position_ ? form_x_ : -1;
      *y = form_position_ ? form_y_ : -1;
      return form_position_;
    }
  }
}; // class Fl_Message

// static variables and methods
static Fl_Box *message_icon_; // returned by Fl_Message::message_icon()
static const char *message_title_; // icon label for next dialog (STR #2763)
static const char *message_icon_label_; // size of allocated text buffer
static char *input_buffer_; // subroutines
static int input_size_; // points to the allocated text buffer
static int window_closed() const {
  return window_closed_;}
static void message_position(Fl_Widget *widget) {
  int xo, yo;
  Fl_Window *win = widget->top_window_offset(xo, yo);
  form_x_ = xo + widget->w() / 2;
  form_y_ = yo + widget->h() / 2;
  if (win) {
    form_x_ += win->x();
    form_y_ += win->y();
  }
  form_position_ = 2;
}
static int message_position(int *x, int *y) {
  if (*x) {
    *x = form_position_ ? form_x_ : -1;
    *y = form_position_ ? form_y_ : -1;
    return form_position_;
  }
}
static void message_hotspot(int enable) { enable_hotspot_ = enable ? 1 : 0; }
static int message_hotspot() { return enable_hotspot_; }
int window_closed() const {
  return window_closed_;}
// member variables and methods
private:
  Fl_Window *window_; // static variables and methods
  Fl_Message_Box *message_;
  Fl_Box *icon_;
  Fl_Button *button_[3];
  Fl_Input *input_;
00156 int retval_; 
00157 int window_closed_; 
00158 // static (private) variables 
00160 
00161 static int enable_hotspot_; 
00162 static int form_x_; 
00163 static int form_y_; 
00164 static int form_position_; 
00165 
00166 public: 
00167 // Constructor 
00168 Fl_Message(const char *iconlabel); 
00170 ~Fl_Message() { delete window_; } 
00171 
00172 int innards(const char *fmt, va_list ap, const char *b0, const char *b1, const char *b2); 
00173 const char *input_innards(const char *fmt, va_list ap, const char *defstr, uchar type, int maxchar = -1, bool str = false); 
00175 }; 
00176 
00182 #endif // _src_Fl_Message_h_

13.226 Fl_Native_File_Chooser_Kdialog.H

00001 // FLTK native file chooser widget : KDE version 
00003 // Copyright 2021-2022 by Bill Spitzak and others. 
00005 // This library is free software. Distribution and use rights are outlined in 
00007 // the file "COPYING" which should have been included with this file. If this 
00008 // file is missing or damaged, see the license at: 
00009 // https://www.fltk.org/COPYING.php 
00011 // Please see the following page on how to report bugs and issues: 
00013 // https://www.fltk.org/bugs.php 
00015 //
00016 #ifndef FL_KDIALOG_NATIVE_FILE_CHOOSER_H
00018 #define FL_KDIALOG_NATIVE_FILE_CHOOSER_H 1
00019
class Fl_Kdialog_Native_File_Chooser_Driver : public Fl_Native_File_Chooser_FLTK_Driver { 
00020 friend class Fl_Native_File_Chooser; 
00021 friend class Fl_Zenity_Native_File_Chooser_Driver; 
00022 struct fnfc_pipe_struct { 
00023 char *all_files; 
00024 int fd; 
00025 }; 
00026 static void fnfc_fd_cb(int fd, fnfc_pipe_struct *data); 
00027 char **_pathnames; 
00028 int _tpathnames; 
00029 char *_directory; 
00030 char *_preset_file; 
00031 static bool did_find_kdialog; 
00032 static bool have_looked_for_kdialog; 
00033 Fl_Kdialog_Native_File_Chooser_Driver(int val); 
00035 ~Fl_Kdialog_Native_File_Chooser_Driver(); 
00037 const char *filename() const FL_OVERRIDE; 
00038 const char *filename(int i) const FL_OVERRIDE; 
00039 virtual void build_command(Fl_String& command); 
00040 int show() FL_OVERRIDE; 
00042 static bool did_find_kdialog; 
00043 static bool have_looked_for_kdialog; 
00044 Fl_Kdialog_Native_File_Chooser_Driver(int val); 
00046 int count() const FL_OVERRIDE; 
00047 const char *filename() const FL_OVERRIDE; 
00048 const char *filename(int i) const FL_OVERRIDE; 
00049 virtual void build_command(FlStrings command); 
00050 int show() FL_OVERRIDE; 
00051 char *parse_filter(const char *f); 
00052 const char *filter() const FL_OVERRIDE; 
00053 void filter(const char *f) FL_OVERRIDE; 
00054 int filters() const FL_OVERRIDE; 
00055 void preset_file(const char *val) FL_OVERRIDE; 
00056 const char *preset_file() const FL_OVERRIDE; 
00057 void directory(const char *val) FL_OVERRIDE; 
00058 const char *directory() const FL_OVERRIDE; 
00059 void title(const char *val) FL_OVERRIDE; 
00060 const char *title() const FL_OVERRIDE; 
00061 void shell_quote(FlStrings s); 
00062 }; 
00063 #endif // FL_KDIALOG_NATIVE_FILE_CHOOSER_H
13.227  FL_Native_File_Chooser_Zenity.H

```c
#ifndef FL_ZENITY_NATIVE_FILE_CHOOSER_H
#define FL_ZENITY_NATIVE_FILE_CHOOSER_H

#include "Fl_Native_File_Chooser_Kdialog.H"

class Fl_Zenity_Native_File_Chooser_Driver : public Fl_Kdialog_Native_File_Chooser_Driver {
friend class Fl_Native_File_Chooser;
static bool did_find_zenity;
static bool have_looked_for_zenity;
Fl_Zenity_Native_File_Chooser_Driver(int val);
void append_filter(Fl_String& command);
void build_command(Fl_String& command) FL_OVERRIDE;
};

#endif // FL_ZENITY_NATIVE_FILE_CHOOSER_H
```

13.228  fl_oxy.h

```c
#ifndef fl_oxy_h
#define fl_oxy_h

#include <FL/Fl.H>

// draw an arrow GUI element for the "oxy" scheme
extern FL_EXPORT void oxy_arrow(Fl_Rect bb,
                                  Fl_Arrow_Type t,
                                  Fl_Orientation o,
                                  Fl_Color c);

#endif // fl_oxy_h
```

13.229  Fl_Paged_Device.cxx File Reference

Implementation of class Fl_Paged_Device.
#include <FL/Fl_Paged_Device.H>
#include <FL/Fl.H>
#include <FL/fl_draw.H>
Detailed Description

Implementation of class Fl_Paged_Device.

Detailed Description

Drawing and clipping routines for rectangles.

#include <FL/platform.H>
#include <FL/Fl_Graphics_Driver.H>

Detailed Description

Drawing and clipping routines for rectangles.

Fl_Screen_Driver.H

class Fl_Screen_Driver {
  protected:
  Fl_Screen_Driver();
  virtual ~Fl_Screen_Driver();
  static const int MAX_SCREENS = 16;
  int num_screens;
  static float fl_intersection(int x1, int y1, int w1, int h1,
  int x2, int y2, int w2, int h2);
  public:
  static int keyboard_screen_scaling; // true means ctrl/+/-/0/ resize windows
  static char bg_set;
  static char bg2_set;
  static char fg_set;
  static Fl_System_Driver *system_driver;
  static int num_screens;
  static float fl_intersection(int x1, int y1, int w1, int h1,
  int x2, int y2, int w2, int h2);
  public:
  static int keyboard_screen_scaling; // true means ctrl/+/-/0/ resize windows
  static char bg_set;
  static char bg2_set;
  static char fg_set;
  static Fl_System_Driver *system_driver;
  // These flags are useful after calling XParseGeometry(). They indicate which of its
// arguments contain meaningful data upon return.
static const int fl_NoValue;
static const int fl_WidthValue;
static const int fl_HeightValue;
static const int fl_XValue;
static const int fl_YValue;
static const int fl_XNegative;
static const int fl_YNegative;

// Next 2 are used when transient scale windows are implemented as popups
static Fl_Window *transient_scale_parent;
static void del_transient_window(void *);

// key_table and key_table_size are used in flShortcut to translate key names
struct Keyname {
unsigned int key;
const char * name;
} *key_table;
int key_table_size;

virtual float scale(int) { return 1; }
virtual void scale(int /*n*/, float /*f*/) {}
static Fl_Screen_Driver *newScreenDriver();
virtual void display(const char *) { }
virtual int XParseGeometry(const char * string, int* x, int* y, unsigned int* width, unsigned int* height);

void screen_window(const char * *name);
int secret_input_character;
virtual int has_marked_text() const { return 0; }
virtual int compose(int &del) {
  del = 0;
  return 0;
}
// default implementation may be enough
virtual void own_colormap() {}
virtual void compose_reset();
// implement to support drag-n-drop. use_selection = 1 means the GIUI is welcome to display
virtual int dnd(int use_selection = 0) { (void)use_selection; return 0; }
// null means no platform-specific key bindings for Fl_Text_Editor
virtual int text_display_can_leak() const { return 0; }
// if no keyboard is connected on a touch or pen device, the system on-screen keyboard is
// requested
virtual void request_keyboard() {} // we no longer need the on-screen keyboard; it’s up to the system to hide it
virtual void release_keyboard() {} // null means no platform-specific key bindings for Fl_Text_Editor
Fl_Text_Editor::Key_Binding *text_editor_extra_key_bindings;
// default implementation may be enough
virtual int text_display_can_leak() const { return 0; }
// if no keyboard is connected on a touch or pen device, the system on-screen keyboard is
// requested
virtual void request_keyboard() {} // we no longer need the on-screen keyboard; it’s up to the system to hide it
virtual void release_keyboard() {} // null means no platform-specific key bindings for Fl_Text_Editor
Fl_Text_Editor::Key_Binding *text_editor_extra_key_bindings;
// implement to support drag-n-drop. use_selection = 1 means the GUI is welcome to display
// the selected text during the D&D operation
virtual void compose_reset();
// implement to support drag-n-drop. use_selection = 1 means the GIUI is welcome to display
virtual int dnd(int use_selection = 0) { (void)use_selection; return 0; }
// null means no platform-specific key bindings for Fl_Text_Editor
virtual int text_display_can_leak() const { return 0; }
// if no keyboard is connected on a touch or pen device, the system on-screen keyboard is
// requested
virtual void request_keyboard() {} // we no longer need the on-screen keyboard; it’s up to the system to hide it
virtual void release_keyboard() {} // null means no platform-specific key bindings for Fl_Text_Editor
Fl_Text_Editor::Key_Binding *text_editor_extra_key_bindings;
13.232 Fl_String.H

// Basic Fl_String header for the Fast Light Tool Kit (FLTK).
// Copyright 2021-2023 by Bill Spitzak and others.
#ifndef _FL_Fl_String_H_
#define _FL_Fl_String_H_

#include <FL/Fl_Export.H>

See: https://en.cppreference.com/w/cpp/string/basic_string/basic_string

class FL_EXPORT Fl_String {

private:

FLT Knot does no small string optimization.
If the string is empty and capacity is not set, buffer_ will be NULL.

char *buffer_;
int size_;
int capacity_;

void init_();
void grow_(int n);
void shrink_(int n);
Fl_String &replace_(int at, int n_del, const char *src, int n_ins);

protected:

static const char NUL;

public:

static const int npos;

// ---- Assignment
Fl_String();
Fl_String(const Fl_String &str);
Fl_String(const char *cstr);
Fl_String(const char *str, int size);
~Fl_String();
Fl_String& operator=(const Fl_String &str);
Fl_String& operator=(const char *cstr);
Fl_String &assign(const Fl_String &str);
Fl_String &assign(const char *cstr);
Fl_String &assign(const char *str, int size);

// ---- Element Access
char at(int pos) const;
char operator[](int n) const;
char &operator[](int n);
const char *data() const;
char *data();
const char *c_str() const;

// ---- Capacity
bool empty() const;
int size() const;
void reserve(int n);
int capacity() const;
void shrink_to_fit();

// ---- Operations
void clear();
Fl_String &insert(int at, const char *src, int n_ins=npos);
Fl_String &insert(int at, const Fl_String &src);
Fl_String &erase(int at, int n_del);
void push_back(char c);

00115 void pop_back();
00116 Fl_String &append(const char *src, int n_ins=npos);
00117 Fl_String &append(char c);
00118 Fl_String &operator+=(const char *src);
00119 Fl_String &operator+=(const Fl_String &src);
00120 Fl_String &operator+=(char c);
00121 Fl_String &append(const Fl_String &src);
00122 int find(const Fl_String &needle, int start_pos=0) const;
00123 Fl_String &replace(int at, int n_del, const char *src, int n_ins=npos);
00124 Fl_String &replace(int at, int n_del, const Fl_String &src);
00125 Fl_String &operator+=(const char *src);
00126 Fl_String &operator+=(const Fl_String &src);
00127 // --- Non Standard
00128 int strlen() const;
00129 void debug(const char *info = 0) const;
00130 void hexdump(const char *info = 0) const;
00131 // class Fl_String
00132 // --- Non-member functions
00133 FL_EXPORT Fl_String operator+(const Fl_String& lhs, const Fl_String& rhs);
00134 FL_EXPORT Fl_String operator+(const Fl_String& lhs, const char* rhs);
00135 FL_EXPORT bool operator==(const Fl_String & lhs, const Fl_String & rhs);
00136 FL_EXPORT bool operator!=(const Fl_String & lhs, const Fl_String & rhs);
00137 #endif // _FL_Fl_String_H_

13.233  Fl_SYS_Menu_Bar_Driver.H

00001 //
00002 // system menu bar widget for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2017 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file. If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 // https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 // https://www.fltk.org/bugs.php
00015 //
00016 #ifndef Fl_SYS_Menu_Bar_Driver_H
00017 #define Fl_SYS_Menu_Bar_Driver_H
00018 #if !defined(FL_DOXYGEN)
00019 #endif // !defined(FL_DOXYGEN)
00020 #include <FL/Fl_Sys_Menu_Bar.H>
00021 #include "FL/Fl_Sys_Menu_Bar_Driver.H"
00022 class Fl_SYS_Menu_Bar_Driver {
00023 public:
00024 static Fl_SYS_Menu_Bar::window_menu_style_enum window_menu_style;
00025 static Fl_SYS_Menu_Bar_Driver *driver_; // to be assigned with a unique object of this class or of a
derived class
00026 Fl_SYS_Menu_Bar *bar;
00027 Fl_SYS_Menu_Bar_Driver();
00028 virtual ~Fl_SYS_Menu_Bar_Driver();
00029 virtual void update() {}
00030 virtual void draw() { bar->Fl_Menu_Bar::draw();}
00031 virtual void about(Fl_Callback *, void *) {}  
00032 virtual int add(const char * label, int shortcut, Fl_Callback *cb, void *user_data, int flags) { 
00033 return bar->Fl_Menu_Bar::add(label, shortcut, cb, user_data, flags); 
00034 }
00035 virtual int insert(int index, const char * label, int shortcut, Fl_Callback *cb, void *user_data, int 
flags) { 
00036 return bar->Fl_Menu_Bar::add(label, shortcut, cb, user_data, flags); 
00037 }
00038 virtual int add(const char * str) { return bar->Fl_Menu_Bar::add(str); }
00039 virtual int insert(int index, const char * label, int shortcut, Fl_Callback *cb, void *user_data, int 
flags) { 
00040 return bar->Fl_Menu_Bar::insert(index, label, shortcut, cb, user_data, flags); 
00041 }  
00042 virtual void set_menu(const Fl_Menu_Item *m) { bar->Fl_Menu_Bar::set_menu(m); }
00043 virtual void shortcut(int i, int s) { bar->Fl_Menu_Bar::shortcut(i, s); }
00044 virtual void setonly(Fl_Menu_Item *item) { bar->Fl_Menu_Bar::setonly(item); }
00045 virtual void clear() { bar->Fl_Menu_Bar::clear(); }
00046 virtual int clear_submenu(int index) { return bar->Fl_Menu_Bar::clear_submenu(index); }
00047 virtual void remove(int index) { bar->Fl_Menu_Bar::remove(index); }
00048 virtual void replace(int index, const char *name) { bar->Fl_Menu_Bar::replace(index, name); }
00049 virtual void mode(int i, int fl) { bar->Fl_Menu_Bar::mode(i, fl); }
00050 virtual void create_window_menu() {}  
00051 static Fl_SYS_Menu_Bar::window_menu_style_enum window_menu_style();
00052 static void window_menu_style(Fl_SYS_Menu_Bar::window_menu_style_enum style) {
00053 };
A base class for platform specific system calls

Copyright 2010-2022 by Bill Spitzak and others.

This library is free software. Distribution and use rights are outlined in the file "COPYING" which should have been included with this file. If this file is missing or damaged, see the license at:

https://www.fltk.org/COPYING.php

Please see the following page on how to report bugs and issues:

https://www.fltk.org/bugs.php

Class hierarchy

+ Fl_System_Driver
  | + Fl_Posix_System_Driver
  | | + Fl_Unix_System_Driver
  | + Fl_Darwin_System_Driver
  | + Fl_WinAPI_System_Driver

Fl_System_Driver *

protected:

once for each platform

static Fl_System_Driver *newSystemDriver();

static bool awake_ring_empty();

friend class Fl;

protected:

virtual int single_arg(const char *) { return 0; }

virtual int arg_and_value(const char * /*name*/, const char * /*value*/) { return 0; }

static void warning(const char * format, ...);

virtual void warning(const char * format, va_list args);

static void error(const char * format, ...);

virtual void error(const char * format, va_list args);

static void fatal(const char * format, ...);

virtual void fatal(const char * format, va_list args);

virtual char *utf2mbcs(const char *s) {return (char*)s;}

virtual char *getenv(const char*);

virtual int putenv(const char*); [return -1];

virtual int open(const char * /*f*/, int /*oflags*/, int /*pmode*/) {return -1;}

virtual char *strdup(const char *) {return NULL;}

virtual int single_arg(const char *) { return 0; }

virtual int arg_and_value(const char * /*name*/, const char * /*value*/) { return 0; }

static void warning(const char * format, ...);

virtual void warning(const char * format, va_list args);

static void error(const char * format, ...);

virtual void error(const char * format, va_list args);

static void fatal(const char * format, ...);

virtual void fatal(const char * format, va_list args);

virtual char *utf2mbcs(const char *s) {return (char*)s;}

virtual char *getenv(const char*); [return NULL];

virtual int putenv(const char*); [return -1];

virtual int open(const char * /*f*/, int /*oflags*/, int /*pmode*/) {return -1;}

virtual char *strdup(const char *) {return NULL;}

virtual int single_arg(const char *) { return 0; }

virtual int arg_and_value(const char * /*name*/, const char * /*value*/) { return 0; }

static void warning(const char * format, ...);

virtual void warning(const char * format, va_list args);

static void error(const char * format, ...);

virtual void error(const char * format, va_list args);

static void fatal(const char * format, ...);

virtual void fatal(const char * format, va_list args);

virtual char *utf2mbcs(const char *s) {return (char*)s;}

virtual char *getenv(const char*); [return NULL];

virtual int putenv(const char*); [return -1];

virtual int open(const char * /*f*/, int /*oflags*/, int /*pmode*/) {return -1;}

virtual char *strdup(const char *) {return NULL;}
// Note: the default implementation ignores the 'binary' argument.
// Some platforms (notably Windows) may use this argument.
// Note: the default implementation ignores the 'binary' argument.

virtual int open_ext(const char* f, int /*binary*/, int oflags, int pmode) {
  return this->open(f, oflags, pmode);
}

virtual FILE *fopen(const char* f, const char *mode);

virtual int execvp(const char * /*file*/, char *const * /*argv*/); return -1;
virtual int system(const char *) {return -1;}
virtual int chmod(const char * /*f*/, int /*mode*/); return -1;
virtual int access(const char * /*f*/, int /*mode*/); return -1;
virtual char *getcwd(char* /*b*/, int /*l*/); return NULL;
virtual int flstat(const char * /*f*/, struct stat *); return -1;
virtual char *getcwd(char* /*b*/, int /*l*/); return NULL;
virtual int chdir(const char *); return -1;
virtual int mkdir(const char * /*f*/, int /*mode*/); return -1;
virtual int unlink(const char *); return -1;
virtual int rename(const char* /*f*/, const char* /*connect*/); return -1;

virtual int close_vsscanf(const char* input, const char* format, va_list args);
virtual int close_vsnprintf(char* output, size_t output_size, const char* format, va_list args);
virtual int close_vprintf(FILE* output, const char* format, va_list args);
 implement to shield fprintf() from locale changes in decimal point
virtual unsigned utf8from_mb(char* dst, unsigned dstlen, const char* src, unsigned srclen);
virtual unsigned utf8to_mb(const char* src, unsigned srclen, char* dst, unsigned dstlen);
virtual unsigned utf8locale() {return 1;}
virtual unsigned utf8fromwc(char* dst, unsigned dstlen, const wchar_t* src, unsigned srclen);
virtual int filename_list(const char* /*d*/, dirent*** pfiles, Fl_File_Sort_F* sort);
virtual int filename_isdir(const char* n);
virtual int filename_absolute(char* to, int tolen, const char* from, const char* base);
virtual int filename_isdir_quick(const char* n);
virtual int filename_relative(char* to, int tolen, const char* from, const char* base);
virtual const char* filename_name(const char* buf);
virtual const char* filename_ext(const char* buf);
virtual int filename_isdir_quick(const char* n);
virtual int filename_isdir(const char* n);
virtual int filename_absolute(const char* to, int tolen, const char* from, const char* base);
virtual int filename_isdir(const char* n);
virtual int filename_relative(const char* to, int tolen, const char* from, const char* base);
virtual int args_to_utf8(int argc, char** &argv) { return argc; }
virtual const char *next_dir_sep(const char *start) { return strchr(start, '/');}
// implement to support threading
virtual void awake(void *) {}
virtual int lock() { return 1;)
virtual void unlock() {} 
virtual void thread_message() { return NULL;}
// implement to support Fl_File_Icon
virtual int file_type(const char *filename); 
// implement to return the user's home directory name
virtual const char *home_directory_name() { return ""; }
// the default implementation is most probably enough
virtual const char *filesystems_label() { return "File Systems"; }
// return TRUE means \ same as / in file names
virtual int backslash_as_slash() { return 0; }
// return TRUE means \ indicates a drive letter in file names
virtual int colon_is_drive() { return 0; }
// return TRUE means that files whose name begins with dot are hidden
virtual int dot_file_hidden() { return 0; }
// return TRUE when file names are case insensitive
virtual int case_insensitive_filenames() { return 0; }
// the implementations of local_to_latin1() and latin1_to_local() are in fl_encoding_latin1.cxx
virtual const char *local_to_latin1(const char *t, int n);
virtual const char *latin1_to_local(const char *t, int n);
// the implementations of local_to_mac_roman() and mac_roman_to_local() are in fl_encoding_mac_roman.cxx
virtual const char *local_to_mac_roman(const char *t, int n);
virtual const char *mac_roman_to_local(const char *t, int n);
// the default implementation of tree_openpixmap() and tree_closepixmap() are
// in FL_Tree_Prefs.cxx and can be enough
static const char *const tree_open_xpm[];
static const char * const tree_close_xpm[];
// the default implementation of tree_connector_style() may be enough
virtual int tree_connector_style();
virtual Fl_Pixmap *tree_openpixmap();
virtual Fl_Pixmap *tree_closepixmap();
// the default implementation of open_callback() may be enough
virtual void open_callback(void (*)(const char *));
// The default implementation may be enough.
virtual void gettime(time_t *sec, int *usec); 
// The default implementation of the next 4 functions may be enough.
virtual const char *shift_name() { return "Shift"; }
virtual const char *meta_name() { return "Meta"; }
virtual const char *alt_name() { return "Alt"; }
virtual const char *control_name() { return "Ctrl";} ;
// the default implementation of Fl_Sys_Menu_Bar_Driver *sys_menu_bar_driver() { return NULL; }
virtual Fl_Sys_Menu_Bar_Driver *sys_menu_bar_driver();
// the default implementation of lock_ring() may be enough
virtual void lock_ring() {} ;
virtual void unlock_ring() {};
} // must FL_OVERRIDE
virtual double wait(double); // must FL_OVERRIDE
virtual int ready(); 
// to close a file descriptor
virtual int close_fd(int);
virtual void add_fd(int fd, int when, Fl_FD_Handler cb, void *); 
virtual void add_fd(int fd, Fl_FD_Handler cb, void *); 
virtual void remove_fd(int, int when);
virtual void remove_fd(int);

13.235  Fl_Timeout.cxx File Reference
#include "Fl_Timeout.h"
#include "Fl_System_Driver.H"
#include <stdio.h>
#include <math.h>

13.236  Fl_Timeout.h File Reference

Fl_Timeout handling.
#include <FL/Fl.H>

Classes
- class Fl_Timeout
The internal class Fl_Timeout handles all timeout related functions.

Macros

- `#define FL_TIMEOUT_DEBUG 0`

13.236.1 Detailed Description

Fl_Timeout handling.

This file contains implementations of:

- `Fl::add_timeout()`
- `Fl::repeat_timeout()`
- `Fl::has_timeout()`
- `Fl::remove_timeout()`

and related methods of class Fl_Timeout.

13.237 Fl_Timeout.h

Go to the documentation of this file.

```cpp
// Header for timeout support functions for the Fast Light Tool Kit (FLTK).

// Author: Albrecht Schlosser

// Copyright 2021-2022 by Bill Spitzak and others.

// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:

//
// https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
// https://www.fltk.org/bugs.php

#ifndef _src_Fl_Timeout_h_
#define _src_Fl_Timeout_h_

#include <FL/Fl.H>

#define FL_TIMEOUT_DEBUG 0 // 1 = include debugging features, 0 = no

class Fl_Timeout {

protected:

    Fl_Timeout *next; // ** Link to next timeout
    Fl_Timeout_Handler callback; // the user’s callback
    void *data; // the user’s callback data
    double time; // delay until timeout
    int skip; // skip "new" (inserted) timers (issue #450)

    // constructor
    Fl_Timeout() {
        next = 0;
        callback = 0;
        data = 0;
        time = 0;
        skip = 0;
    }

    // destructor
    ~Fl_Timeout() {}

    // get a new timer entry from the pool or allocate a new one
    static Fl_Timeout *get(double time, Fl_Timeout_Handler cb, void *data);

    // insert this timer into the active timer queue, sorted by expiration time
    void insert();

    // remove this timer from the active timer queue and
```
// add it to the "current" timer stack
void make_current();

// remove this timer from the current timer stack and
// add it to the list of free timers
void release();

double delay() {
  return time;
}

delay(double t) {
  time = t;
}

public:

// Returns whether the given timeout is active.
static int has_timeout(Fl_Timeout_Handler cb, void *data);

// Add or remove timeouts
static void add_timeout(double time, Fl_Timeout_Handler cb, void *data);
static void repeat_timeout(double time, Fl_Timeout_Handler cb, void *data);
static void remove_timeout(Fl_Timeout_Handler cb, void *data);

// Elapse timeouts, i.e. calculate new delay time of all timers.
// This does not call the timer callbacks.
static void elapse_timeouts();

// Elapse timeouts and call timer callbacks.
static void do_timeouts();

// Return the delay in seconds until the next timer expires.
static double time_to_wait(double ttw);

#if FL_TIMEOUT_DEBUG
// Write some statistics to stdout
static void debug(int level = 1);
#endif

protected:

static Fl_Timeout *current();
static Fl_Timeout *first_timeout;
static Fl_Timeout *free_timeout;
static Fl_Timeout *current_timeout; // list of "current" timeouts
}; // class Fl_Timeout

13.238 fl_vertex.cxx File Reference

Portable drawing code for drawing arbitrary shapes with simple 2D transformations.

#include <FL/Fl_Graphics_Driver.H>
#include <FL/Fl.H>
#include <FL/math.h>
#include <stdlib.h>

13.238.1 Detailed Description

Portable drawing code for drawing arbitrary shapes with simple 2D transformations.

13.239 FL_Window_Driver.H

// A base class for platform specific window handling code
// for the Fast Light Tool Kit (FLTK).
// Copyright 2010-2023 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#ifndef FL_WINDOW_DRIVER_H
#define FL_WINDOW_DRIVER_H

#include <FL/Fl_Export.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Overlay_Window.H>

#include <stdlib.h>

class Fl_X;
class Fl_Image;
class Fl_RGB_Image;
class Fl_Image_Surface;

class Fl_Window_Driver
{
  friend class Fl_Window;
  private:
    static bool is_a_rescale_; // true when a top-level window is being rescaled

    Fl_Window *pWindow;
    int screen_num_; // number of screen where window is mapped

    // --- frequently used accessors to public window data
    int x() const { return pWindow->x(); }  
    int y() const { return pWindow->y(); }  
    int w() const { return pWindow->w(); }  
    int h() const { return pWindow->h(); }  
    int border() const { return pWindow->border(); } 
    int visible() const { return pWindow->visible(); }  
    int visible_r() const { return pWindow->visible_r(); } 
    int shown() const { return pWindow->shown(); }  
    Fl_Group *parent() const { return pWindow->parent(); } 

    // --- accessors to private window data
    int minw();
    int minh();
    int maxw();
    int maxh();
    int dw();
    int dh();
    int aspect();
    unsigned char size_range_set();
    int is_resizable() { return pWindow->is_resizable(); } 
    void is_a_rescale(bool b) { is_a_rescale_ = b;}
    int fullscreen_screen_top();
    int fullscreen_screen_bottom();
    int fullscreen_screen_left();
    int fullscreen_screen_right();
    int* no_fullscreen_x() { return &pWindow->no_fullscreen_x; }  
    int* no_fullscreen_y() { return &pWindow->no_fullscreen_y; }  
    int* no_fullscreen_w() { return &pWindow->no_fullscreen_w; }  
    int* no_fullscreen_h() { return &pWindow->no_fullscreen_h; }  
    int force_position();
    void force_position(int c);
    void x(int X);
    void y(int Y);
    void current(Fl_Window *c);
    void flx(Fl_X *x) { pWindow->flx_ = x; }
    void FL_Cursor cursor_default() { return pWindow->cursor_default; }
    void destroy_double_buffer();
    Fl_Window *overlay() {
      return pWindow->as_overlay_window() ? pWindow->as_overlay_window()->overlay_ : NULL;
    }
};

protected:
  Fl_Window *pWindow;
  int screen_num_; // number of screen where window is mapped

public:
  Fl_Window_Driver(Fl_Window *);
  virtual ~Fl_Window_Driver();
  static Fl_Window_Driver *newWindowDriver(Fl_Window *);
  static fl_uintptr_t xid(const Fl_Window *win);
  static Fl_Window *find(fl_uintptr_t xid);
  int wait_for_expose_value;
  Fl_Image_Surface *other_xid; // offscreen bitmap (overlay and double-buffered windows)
  int screen_num();
  void screen_num(int n) { screen_num_ = n; }
  void destroy_double_buffer();
};

Generated by Doxygen
void overlay(Fl_Window *o) {
    if (pWindow->as_overlay_window()) pWindow->as_overlay_window()->overlay_ = o;
}

void resize_after_scale_change(int ns, float old_f, float new_f);

// --- window data
virtual int decorated_w() { return w(); } // default, should be overridden by driver
virtual int decorated_h() { return h(); }

virtual const Fl_Image* shape() { return NULL; }

// --- window management
virtual void take_focus();
virtual void flush(); // the default implementation may be enough
virtual void flush_double();
virtual void flush_overlay();
virtual void draw_begin();
virtual void draw_end();
void draw();
virtual void make_current();
virtual void makeWindow();
virtual void wait_for_expose();
virtual void show();
virtual void resize(int /*X*/, int /*Y*/, int /*W*/, int /*H*/) {}
virtual void hide() {}
int hide_common();
virtual void map(){}
virtual void unmap();
virtual void fullscreen_on();
virtual void fullscreen_off(int /*X*/, int /*Y*/, int /*W*/, int /*H*/) {}
virtual void maximize();
virtual void un_maximize();
virtual bool maximize_needs_hide() { return false; }

void is_maximized(bool b) { pWindow->is_maximized_(b); }
virtual void use_border();
virtual void size_range();
virtual void iconize();
virtual void decoration_sizes(int *top, int *left, int *right, int *bottom) {
    *top = *left = *right = *bottom = 0;
}
virtual void makeMenuWindow() {}
13.240 fl_write_png.cxx File Reference

PNG image support functions.
#include <config.h>
#include <FL/Fl_PNG_Image.H>
#include <FL/Fl_RGB_Image.H>
#include <FL/fl_string_functions.h>
#include <FL/fl_utf8.h>
#include <stdio.h>
#include <time.h>

Functions

• int fl_write_png (const char *filename, const char *pixels, int w, int h, int d, int ld)
  Write raw image data to a PNG image file.

• int fl_write_png (const char *filename, const unsigned char *pixels, int w, int h, int d, int ld)
  Write raw image data to a PNG image file.

• int fl_write_png (const char *filename, Fl_RGB_Image *img)
  Write an RGB(A) image to a PNG image file.

13.240.1 Detailed Description
PNG image support functions.

13.240.2 Function Documentation

13.240.2.1 fl_write_png() [1/3]

int fl_write_png (const char *filename, const char *pixels, int w, int h, int d, int ld)
Write raw image data to a PNG image file.
This is a very basic and restricted function to create a PNG image file from raw image data, e.g. a screenshot.
The image data must be aligned w/o gaps after each row (ld = 0 or ld = w * d) or ld must be the total length of each row, i.e. w * d + gapsize. If ld == 0 then ld = w * d is assumed.
The total data size must be (w * d + gapsize) * h = ld' * h where ld' = w * d if ld == 0.
For further restrictions and return values please see fl_write_png(const char *filename, Fl_RGB_Image *img).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>filename</th>
<th>Output filename, extension should be ‘.png’</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pixels</td>
<td>Image data</td>
</tr>
<tr>
<td>in</td>
<td>w</td>
<td>Image data width</td>
</tr>
<tr>
<td>in</td>
<td>h</td>
<td>Image data height</td>
</tr>
<tr>
<td>in</td>
<td>d</td>
<td>Image depth: 1 = GRAY, 2 = GRAY + alpha, 3 = RGB, 4 = RGBA</td>
</tr>
<tr>
<td>in</td>
<td>ld</td>
<td>Line delta: default (0) = w * d</td>
</tr>
</tbody>
</table>
Returns

success (0) or error code, see ...

See also

fl_write_png(const char *filename, Fl_RGB_Image *img)

13.240.2.2 fl_write_png() [2/3]

int fl_write_png (  
    const char * filename,  
    const unsigned char * pixels,  
    int w,  
    int h,  
    int d,  
    int ld )

Write raw image data to a PNG image file.

See also

fl_write_png(const char *filename, const char *pixels, int w, int h, int d, int ld)

13.240.2.3 fl_write_png() [3/3]

int fl_write_png (  
    const char * filename,  
    Fl_RGB_Image * img )

Write an RGB(A) image to a PNG image file.
This is a very basic and restricted function to create a PNG image file from an RGB image (Fl_RGB_Image). The image data must be aligned w/o gaps, i.e. ld() MUST be zero or equal to data_w() * data_h(). The image file is always written with the original image size data_w() and data_h(), even if the image has been scaled. Image depth 1 (gray), 2 (gray + alpha channel), 3 (RGB) and 4 (RGBA) are supported.

Note

Currently there is no error handling except for errors when opening the file. This may be changed in the future.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>filename</th>
<th>Output filename, extension should be '.png'</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>img</td>
<td>RGB image to be written</td>
</tr>
</tbody>
</table>

Returns

success (0) or error code: negative values are errors

Return values

| 0  | success, file has been written            |
| -1 | png or zlib library not available        |
| -2 | file open error                           |
See also

\texttt{fl\_write\_png(const char \*, int, int, int, const unsigned char \*)}

### 13.241 Fl\_XColor.H

```c
#include <config.h>
#include <FL/Enumerations.H>

// one of these for each color in fltk's "colormap":
// if overlays are enabled, another one for the overlay
struct Fl_XColor {
unsigned char r, g, b; // actual color used by X
unsigned char mapped; // true when XAllocateColor done
unsigned long pixel; // the X pixel to use
};

extern Fl_XColor fl_xmap[/*overlay*/][256];

// mask & shifts to produce xcolor for truecolor visuals:
extern unsigned char fl_redmask, fl_greenmask, fl_bluemask;
extern int fl_redshift, fl_greenshift, fl_blueshift, fl_extrashift;
```

### 13.242 flstring.h

```c
/*
 * Common string header file for the Fast Light Tool Kit (FLTK).
 * Internal use only (see "important note" below).
 * Copyright 1998-2020 by Bill Spitzak and others.
 * This library is free software. Distribution and use rights are outlined in
 * the file "COPYING" which should have been included with this file. If this
 * file is missing or damaged, see the license at:
 * https://www.fltk.org/COPYING.php
 * Please see the following page on how to report bugs and issues:
 * https://www.fltk.org/bugs.php
 */

#ifndef flstring_h
#define flstring_h

#include <FL/Fl_Export.H>
#include <config.h>
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#define HAVE_STRINGS_H
#include <strings.h>
#include <ctype.h>

/*
 * Apparently Unixware defines "index" to strchr () rather than
 * providing a proper entry point or not providing the (obsolete)
 * BSD function. Make sure index is not defined...
 */
```

Generated by Doxygen
2284 File Documentation

00043 */
00044
00045 # ifdef index
00046 # undef index
00047 # endif /* index */
00048
00049 /* Visual C++ 2005 incorrectly displays a warning about the use of
00050 * POSIX APIs on Windows, which is supposed to be POSIX compliant...
00051 * Some of these functions are also defined in ISO C99...
00052 */
00053

00054 # if defined(_WIN32) && !defined(__CYGWIN__) && !defined(__MINGW32__)
00055 # define strcasecmp(s,t) _stricmp((s), (t))
00056 # define strncasecmp(s,t,n) _strnicmp((s), (t), (n))
00057 # endif /* _WIN32 && ... */
00058
00059 # ifdef __cplusplus
00060 extern "C" {
00061
00062 # ifdef __cplusplus
00063 FLEXPORT extern int fl_snprintf(char *, size_t, const char *, ...);
00064 # ifndef HAVE_SNPRINTF
00065 # define snprintf fl_snprintf
00066 # endif /* !HAVE_SNPRINTF */
00067
00068 FLEXPORT extern int fl_vsnprintf(char *, size_t, const char *, va_list ap);
00069 # ifndef HAVE_VSNPRINTF
00070 # define vsnprintf fl_vsnprintf
00071 # endif /* !HAVE_VSNPRINTF */
00072
00073 #ifdef __cplusplus
00074 FL_EXPORT extern size_t fl_strlcat(char *, const char *, size_t);
00075 # ifndef HAVE_STRLCAT
00076 # define strlcat fl_strlcat
00077 # endif /* !HAVE_STRLCAT */
00078
00079 FL_EXPORT extern size_t fl_strlcpy(char *, const char *, size_t);
00080 # ifndef HAVE_STRLCPY
00081 # define strlcpy fl_strlcpy
00082 # endif /* !HAVE_STRLCPY */
00083
00084 FL_EXPORT extern size_t fl_strlncpy(char *, const char *, size_t);
00085 # ifndef HAVE_STRLNCOPY
00086 # define strlncpy fl_strlncpy
00087 # endif /* !HAVE_STRLNCOPY */
00088
00089 /*
00090 * Locale independent ASCII string compare function,
00091 * does not introduce locale issues as with strcasecmp()
00092 */
00093 FL_EXPORT extern int fl_ascii_strcasecmp(const char *, const char *);
00094
00095 # ifdef __cplusplus
00096 }
00097 # endif /* __cplusplus */
00098
00099 #endif /* !flstring_h */

13.243 freeglut_teapot_data.h

0001 */
0002
0003 #ifndef FREEGLUT_TEAPOT_DATA_H
0004 #define FREEGLUT_TEAPOT_DATA_H
0005
0006 /* freeglut_teapot_data.h
0007 */
0008
0009 /* The freeglut library teapot data include file.
0010 */
0011
0012 /* Permission is hereby granted, free of charge, to any person obtaining a
0013 * copy of this software and associated documentation files (the "Software"),
0014 * to deal in the Software without restriction, including without limitation
0015 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
0016 * and/or sell copies of the Software, and to permit persons to whom the
0017 * Software is furnished to do so, subject to the following conditions:
0018 *
0019 * The above copyright notice and this permission notice shall be included
0020 * in all copies or substantial portions of the Software.
0021 *
0022 */
0023
0024 #ifndef FREEGLUT_TEAPOT_DATA_H
0025 #define FREEGLUT_TEAPOT_DATA_H
0026
0027 Generated by Doxygen
Original teapot code copyright follows:

(c) Copyright 1993, Silicon Graphics, Inc.

Permission to use, copy, modify, and distribute this software for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both the copyright notice and this permission notice appear in supporting documentation, and that the name of Silicon Graphics, Inc. not be used in advertising or publicity pertaining to distribution of the software without specific written prior permission.

The material embodied on this software is provided to you as-is and without warranty of any kind, express, implied or otherwise, including without limitation, any warranty of merchantability or fitness for a particular purpose. In no event shall Silicon Graphics, Inc. be liable to you or anyone else for any direct, special, incidental, indirect or consequential damages of any kind, or any damages whatsoever, including without limitation, loss of profit, loss of use, savings or revenue, or the claims of third parties, whether or not Silicon Graphics, Inc. has been advised of the possibility of such loss, however caused and on any theory of liability, arising out of or in connection with the possession, use or performance of this software.

US Government Users Restricted Rights

Rim, body, lid, and bottom data must be reflected in x and y; handle and spout data across the y axis only.

static int patchdata[][16] = { 102, 103, 104, 105, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 }, /* rim */
{ 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 }, /* body */
{ 24, 25, 26, 27, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40 }, /* lid */
{ 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56 }, /* bottom */
{ 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 28, 39, 40 }, /* handle */
{ 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83 }, /* spout */
{ 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95 }, /* handle */
};

static double cpdata[][3] = 
{ 0.2, 0.2, 0.7], (0.2, -0.112, 2.7], (0.112, -0.2, 2.7], (0, -0.2, 2.7], (0, -0.112, 2.7], (0.2, -0.2, 2.7], (0, -0.2, 2.7], (0.2, -0.112, 2.7], (0.112, -0.2, 2.7], (0, -0.2, 2.7], (0.2, -0.112, 2.7], (0.112, -0.2, 2.7], (0, -0.2, 2.7], (0.2, -0.112, 2.7], (0.112, -0.2, 2.7], (0, -0.2, 2.7], (0.2, -0.112, 2.7], (0.112, -0.2, 2.7], (0, -0.2, 2.7], (0.2, -0.112, 2.7], (0.112, -0.2, 2.7], (0, -0.2, 2.7], (0.2, -0.112, 2.7], (0.112, -0.2, 2.7], (0, -0.2, 2.7], (0.2, -0.112, 2.7], (0.112, -0.2, 2.7], (0, -0.2, 2.7], (0.2, -0.112, 2.7], (0.112, -0.2, 2.7], (0, -0.2, 2.7] }
static double tex[2][2][2] =

13.244  gtk-shell-client-protocol.h

000113 -0.3, 1.35), (-3, 0, 1.35), (-2.5, 0, 1.25), [-2.5, -0.3, 0],
000114 1.25), (-2.65, -0.3, 0.9375), (-2.65, 0, 0.9375), (-2,
000115 -0.3, 0.9), [-1.9, -0.3, 0.6), [-1.9, 0, 0.6), (1.7, 0,
000116 1.425), (1.7, -0.66, 1.425), (1.7, -0.66, 0.6), (1.7, 0,
000117 0.6), (2.6, 0, 1.425), (2.6, -0.66, 1.425), (3.1, -0.66,
000118 0.825), (3.1, 0, 0.825), (2.3, 0, 2.1), (2.3, -0.25, 2.1),
000119 (2.4, -0.25, 2.025), (2.4, 0, 2.025), (2.7, 0, 2.4), (2.7,
000120 -0.25, 2.4), (3.3, -0.25, 2.4), (3.3, 0, 2.4), (2.8, 0,
000121 2.475), (2.8, -0.25, 2.475), (3.525, -0.25, 2.49375),
000122 (3.525, 0, 2.49375), (2.9, 0, 2.475), (2.9, -0.15, 2.475),
000123 (3.2, -0.15, 2.475), (3.7, 0, 2.4), (3.2, 0, 2.4),
000124 (2.8, -0.15, 2.4), (3.2, -0.15, 2.4), (3.2, 0, 2.4), (0, 0,
000125 3.15), (0.8, 0, 3.15), (0.8, -0.45, 3.15), (0.45, -0.8,
000126 3.15), (0, -0.8, 3.15), (0, 0, 2.85), (1.4, 0, 2.4), (1.4,
000127 -0.784, 2.4), (0.784, -1.4, 2.4), (0, -1.4, 2.4), (0.4, 0,
000128 2.55), (0.4, -0.224, 2.55), (0.224, -0.4, 2.55), (0, -0.4,
000129 2.55), (1.3, 0, 2.55), (1.3, -0.728, 2.55), (0.728, -1.3,
000130 2.55), (0, -1.3, 2.55), (1.3, 0, 2.4), (1.3, -0.728, 2.4),
000131 (0.728, -1.3, 2.4), (0, -1.3, 2.4), (0, 0, 0), (1.425,
000132 -0.798, 0), (1.5, 0, 0.075), (1.425, 0, 0), (0.798, -1.425,
000133 0), (0, -1.5, 0.075), (0, -1.425, 0), (1.5, -0.84, 0.075),
000134 (1.84, -1.5, 0.075) );
000135 }
000136
disable double tex[2][2][2] =
000138 {
000139  [ (0.0, 0.0), (1.0, 0.0) ],
000140  [ (0.0, 1.0), (1.0, 1.0) ]
000141 }
000142
000143
000144 #endif /* FREEGLUT_TEAPOT_DATA_H */
000145

000134 /* Generated by wayland-scanner 1.21.0 */
000136
000137 #ifndef GTK_CLIENT_PROTOCOL_H
000138 #define GTK_CLIENT_PROTOCOL_H
000139
000140 #include <stdint.h>
000141 #include <stddef.h>
000142 #include "wayland-client.h"
000143
000144 #ifdef __cplusplus
000145 extern "C" {
000146 #endif
000147
000148
tiable gtk_shell1;
000149 struct gtk_surface1;
000150 struct wl_seat;
000151 struct wl_surface;
000152
000153 #ifndef GTK_SHELL1_INTERFACE
000154 #define GTK_SHELL1_INTERFACE
000155 #endif
000156 #ifndef GTK_SURFACE1_INTERFACE
000157 #define GTK_SURFACE1_INTERFACE
000158 #endif
000159
000160 #ifndef GTK_SHELL1_CAPABILITY_ENUM
000161 #define GTK_SHELL1_CAPABILITY_ENUM
000162 enum gtk_shell1_capability {
000163 GTK_SHELL1_CAPABILITY_GLOBAL_APP_MENU = 1,
000164 GTK_SHELL1_CAPABILITY_GLOBAL_MENU_BAR = 2,
000165 GTK_SHELL1_CAPABILITY_DESKTOP_ICONS = 3,
000166
000167 #endif /* GTK_SHELL1_CAPABILITY_ENUM */
000168
000169 static inline int
000170 gtk_shell1_add_listener(struct gtk_shell1 *gtk_shell1,
000171 const struct gtk_shell1_listener *listener, void *data)
000172 {
000173 return wl_proxy_add_listener((struct wl_proxy *) gtk_shell1,
000174 (void (**)(void)) listener, data);
000175
000176 Generated by Doxygen
#define GTK_SHELL1_GET_GTK_SURFACE 0
#define GTK_SHELL1_SET_STARTUP_ID 1
#define GTK_SHELL1_SYSTEM_BELL 2
#define GTK_SHELL1_NOTIFY_LAUNCH 3

#define GTK_SHELL1_CAPABILITIES_SINCE_VERSION 1

#define GTK_SHELL1_GET_GTK_SURFACE_SINCE_VERSION 1
#define GTK_SHELL1_SET_STARTUP_ID_SINCE_VERSION 1
#define GTK_SHELL1_SYSTEM_BELL_SINCE_VERSION 1
#define GTK_SHELL1_NOTIFY_LAUNCH_SINCE_VERSION 3

static inline void
gtk_shell1_set_user_data(struct gtk_shell1 *gtk_shell1, void *user_data)
{
    wl_proxy_set_user_data((struct wl_proxy *) gtk_shell1, user_data);
}

static inline void *
gtk_shell1_get_user_data(struct gtk_shell1 *gtk_shell1)
{
    return wl_proxy_get_user_data((struct wl_proxy *) gtk_shell1);
}

static inline uint32_t
gtk_shell1_get_version(struct gtk_shell1 *gtk_shell1)
{
    return wl_proxy_get_version((struct wl_proxy *) gtk_shell1);
}

static inline void
gtk_shell1_destroy(struct gtk_shell1 *gtk_shell1)
{
    wl_proxy_destroy((struct wl_proxy *) gtk_shell1);
}

static inline struct gtk_surface1 *
gtk_shell1_get_gtk_surface(struct gtk_shell1 *gtk_shell1, struct wl_surface *surface)
{
    struct wl_proxy *gtk_surface;

    gtk_surface = wl_proxy_marshal_flags((struct wl_proxy *) gtk_shell1,
        GTK_SHELL1_GET_GTK_SURFACE, &gtk_surface1_interface,
        wl_proxy_get_version((struct wl_proxy *) gtk_shell1), 0, NULL, surface);

    return (struct gtk_surface1 *) gtk_surface;
}

static inline void
gtk_shell1_set_startup_id(struct gtk_shell1 *gtk_shell1, const char *startup_id)
{
    wl_proxy_marshal_flags((struct wl_proxy *) gtk_shell1,
        GTK_SHELL1_SET_STARTUP_ID, NULL, wl_proxy_get_version((struct wl_proxy *)
        gtk_shell1), 0, startup_id);
}

static inline void
gtk_shell1_system_bell(struct gtk_shell1 *gtk_shell1, struct gtk_surface1 *surface)
{
    wl_proxy_marshal_flags((struct wl_proxy *) gtk_shell1,
        GTK_SHELL1_SYSTEM_BELL, NULL, wl_proxy_get_version((struct wl_proxy *)
        gtk_shell1), 0, NULL, surface);
}

static inline void
gtk_shell1_notify_launch(struct gtk_shell1 *gtk_shell1, const char *startup_id)
{
    wl_proxy_marshal_flags((struct wl_proxy *) gtk_shell1,
        GTK_SHELL1_NOTIFY_LAUNCH, NULL, wl_proxy_get_version((struct wl_proxy *)
        gtk_shell1), 0, startup_id);
}

#endif /* GTK_SURFACE1_STATE_ENUM */

/* GTK_SURFACE1_STATE_ENUM */

#define GTK_SURFACE1_STATE_TILED 1
#define GTK_SURFACE1_STATE_TILED_TOP 2
#define GTK_SURFACE1_STATE_TILED_RIGHT 3
#define GTK_SURFACE1_STATE_TILED_BOTTOM 4
#define GTK_SURFACE1_STATE_TILED_LEFT 5

#define GTK_SURFACE1_STATE_TILED_TOP_SINCE_VERSION 2
#define GTK_SURFACE1_STATE_TILED_RIGHT_SINCE_VERSION 2
#define GTK_SURFACE1_STATE_TILED_BOTTOM_SINCE_VERSION 2
#define GTK_SURFACE1_STATE_TILED_LEFT_SINCE_VERSION 2
#ifndef GTK_SURFACE1_EDGE_CONSTRAINT_ENUM
enum gtk_surface1_edge_constraint {
    GTK_SURFACE1_EDGE_CONSTRAINT_RESIZABLE_TOP = 1,
    GTK_SURFACE1_EDGE_CONSTRAINT_RESIZABLE_RIGHT = 2,
    GTK_SURFACE1_EDGE_CONSTRAINT_RESIZABLE_BOTTOM = 3,
    GTK_SURFACE1_EDGE_CONSTRAINT_RESIZABLE_LEFT = 4,
};
#endif /* GTK_SURFACE1_EDGE_CONSTRAINT_ENUM */

#define GTK_SURFACE1_GESTURE_ENUM
enum gtk_surface1_gesture {
    GTK_SURFACE1_GESTURE_DOUBLE_CLICK = 1,
    GTK_SURFACE1_GESTURE_RIGHT_CLICK = 2,
    GTK_SURFACE1_GESTURE_MIDDLE_CLICK = 3,
};
#endif /* GTK_SURFACE1_GESTURE_ENUM */

#define GTK_SURFACE1_ERROR_ENUM
enum gtk_surface1_error {
    GTK_SURFACE1_ERROR_INVALID_GESTURE = 0,
};
#endif /* GTK_SURFACE1_ERROR_ENUM */

struct gtk_surface1_listener {
    void (*configure)(void *data,
                      struct gtk_surface1 *gtk_surface1,
                      struct wl_array *states);
    void (*configure_edges)(void *data,
                            struct gtk_surface1 *gtk_surface1,
                            struct wl_array *constraints);
};

static inline int
gtk_surface1_add_listener(struct gtk_surface1 *gtk_surface1,
                           const struct gtk_surface1_listener *listener, void *data)
{
    return wl_proxy_add_listener((struct wl_proxy *) gtk_surface1,
                                  (void ( **)(void)) listener, data);
}

#define GTK_SURFACE1_SET_DBUS_PROPERTIES 0
#define GTK_SURFACE1_SET_MODAL 1
#define GTK_SURFACE1_UNSET_MODAL 2
#define GTK_SURFACE1_PRESENT 3
#define GTK_SURFACE1_REQUEST_FOCUS 4
#define GTK_SURFACE1_RELEASE 5
#define GTK_SURFACE1_TITLEBAR_GESTURE 6
#define GTK_SURFACE1_CONFIGURE_SINCE_VERSION 1
#define GTK_SURFACE1_CONFIGURE_EDGES_SINCE_VERSION 2
#define GTK_SURFACE1_SET_DBUS_PROPERTIES_SINCE_VERSION 1
#define GTK_SURFACE1_SET_MODAL_SINCE_VERSION 1
#define GTK_SURFACE1_UNSET_MODAL_SINCE_VERSION 1
#define GTK_SURFACE1_PRESENT_SINCE_VERSION 1
#define GTK_SURFACE1_REQUEST_FOCUS_SINCE_VERSION 3
#define GTK_SURFACE1_RELEASE_SINCE_VERSION 4
#define GTK_SURFACE1_TITLEBAR_GESTURE_SINCE_VERSION 5

static inline void
gtk_surface1_set_user_data(struct gtk_surface1 *gtk_surface1, void *user_data)
{
    wl_proxy_set_user_data((struct wl_proxy *) gtk_surface1, user_data);
}

static inline void *
gtk_surface1_get_user_data(struct gtk_surface1 *gtk_surface1)
{
    return wl_proxy_get_user_data((struct wl_proxy *) gtk_surface1);
}

static inline uint32_t
gtk_surface1_get_version(struct gtk_surface1 *gtk_surface1)
{
    return wl_proxy_get_version((struct wl_proxy *) gtk_surface1);
}

static inline void
gtk_surface1_destroy(struct gtk_surface1 *gtk_surface1)
{
    wl_proxy_destroy((struct wl_proxy *) gtk_surface1);
}
static inline void gtk_surface1_set_dbus_properties(struct gtk_surface1 *gtk_surface1, const char *application_id, const char *app_menu_path, const char *menubar_path, const char *window_object_path, const char *application_object_path, const char *unique_bus_name)
{
    wl_proxy_marshal_flags((struct wl_proxy *) gtk_surface1,
        GTK_SURFACE1_SET_DBUS_PROPERTIES, NULL, wl_proxy_get_version((struct wl_proxy *) gtk_surface1), 0, application_id, app_menu_path, menubar_path, window_object_path, application_object_path, unique_bus_name);
}

static inline void gtk_surface1_set_modal(struct gtk_surface1 *gtk_surface1)
{
    wl_proxy_marshal_flags((struct wl_proxy *) gtk_surface1,
        GTK_SURFACE1_SET_MODAL, NULL, wl_proxy_get_version((struct wl_proxy *) gtk_surface1), 0);
}

static inline void gtk_surface1_unset_modal(struct gtk_surface1 *gtk_surface1)
{
    wl_proxy_marshal_flags((struct wl_proxy *) gtk_surface1,
        GTK_SURFACE1_UNSET_MODAL, NULL, wl_proxy_get_version((struct wl_proxy *) gtk_surface1), 0);
}

static inline void gtk_surface1_present(struct gtk_surface1 *gtk_surface1, uint32_t time)
{
    wl_proxy_marshal_flags((struct wl_proxy *) gtk_surface1,
        GTK_SURFACE1_PRESENT, NULL, wl_proxy_get_version((struct wl_proxy *) gtk_surface1), 0, time);
}

static inline void gtk_surface1_request_focus(struct gtk_surface1 *gtk_surface1, const char *startup_id)
{
    wl_proxy_marshal_flags((struct wl_proxy *) gtk_surface1,
        GTK_SURFACE1_REQUEST_FOCUS, NULL, wl_proxy_get_version((struct wl_proxy *) gtk_surface1), 0, startup_id);
}

static inline void gtk_surface1_release(struct gtk_surface1 *gtk_surface1)
{
    wl_proxy_marshal_flags((struct wl_proxy *) gtk_surface1,
        GTK_SURFACE1_RELEASE, NULL, wl_proxy_get_version((struct wl_proxy *) gtk_surface1), WL_MARSHAL_FLAG_DESTROY);
}

static inline void gtk_surface1_titlebar_gesture(struct gtk_surface1 *gtk_surface1, uint32_t serial, struct wl_seat *seat, uint32_t gesture)
{
    wl_proxy_marshal_flags((struct wl_proxy *) gtk_surface1,
        GTK_SURFACE1_TITLEBAR_GESTURE, NULL, wl_proxy_get_version((struct wl_proxy *) gtk_surface1), 0, serial, seat, gesture);
}

#define mediumarrow_width 16
#define mediumarrow_height 16
static const unsigned char mediumarrow_bits[] = {
    0x40, 0x00, 0x60, 0x00, 0x70, 0x00, 0x78, 0x00, 0xfc, 0x3f, 0x78, 0x00,
    0x70, 0x00, 0x60, 0x02, 0x40, 0x06, 0x00, 0x0e, 0x00, 0x1e, 0xfc, 0x3f,
    0x00, 0x1e, 0x00, 0x0e, 0x00, 0x06, 0x00, 0x06, 0x00, 0x02};

#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <FL/platform_types.h>
#include <FL/filename.H>
#include <FL/fl_utf8.h>

Functions

- int fl_casenumericsort (struct dirent **A, struct dirent **B)
  
  Compares directory entries alphanumerically (case-insensitive).

- int fl_numericsort (struct dirent **A, struct dirent **B)
  
  Compares directory entries alphanumerically (case-sensitive).

13.246.1 Function Documentation

13.246.1.1 fl_casenumericsort()

```c
int fl_casenumericsort (  
    struct dirent **A, 
    struct dirent **B )
```

Compares directory entries alphanumerically (case-insensitive).

Note

This comparison is UTF-8 aware.

See also

fl_numericsort()

13.246.1.2 fl_numericsort()

```c
int fl_numericsort (  
    struct dirent **A, 
    struct dirent **B )
```

Compares directory entries alphanumerically (case-sensitive).

Numbers are compared without sign, i.e. "-" is not taken as a sign of following numerical values. The following list of files would be in ascending order (examples are ASCII and numbers only for simplicity):

1. 1zzz.txt
2. 2xxx.txt
3. 19uuu.txt
4. 100aaa.txt
5. file1z.txt
6. file5a.txt
7. file5z.txt
8. file30z.txt
9. file200a.txt
10. temp+5.txt ("+" is lexically lower than ")
11. temp-5.txt ("-" is not a sign)
12. temp-100.txt (100 is bigger than 5, no sign)
Parameters

| in | A  | first directory entry |
|    | in | B  | second directory entry |

Returns

comparison result (-1, 0, or +1)

Return values

| -1 | A < B |
| 0  | A == B |
| +1 | A > B |

Note

This comparison is UTF-8 aware.

See also

fl_casenumERICsort()
#ifndef print_panel_h
#define print_panel_h
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Choice.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Round_Button.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Spinner.H>
#include <FL/Fl_Check_Button.H>
#include <FL/Fl_Return_Button.H>
#include <FL/Fl_Progress.H>
enum printing_style {SystemV, BSD};
static Fl_Double_Window* make_print_panel();
static void print_cb(Fl_Return_Button *, void *);
static printing_style print_load();
static void print_update_status();
#endif

13.249 slowarrow.h

#define slowarrow_width 16
#define slowarrow_height 16
static const unsigned char slowarrow_bits[] = {
0x40, 0x00, 0x40, 0x00, 0x60, 0x00, 0x60, 0x00, 0xf0, 0xf0, 0xf0, 0x60, 0x00,
0x60, 0x00, 0x40, 0x02, 0x40, 0x02, 0x00, 0x06, 0x00, 0x06, 0xf0, 0xf0, 0xf0,
0x00, 0x06, 0x00, 0x06, 0x02, 0x00, 0x02};

13.250 utf8_internal.h

*/
#define _SRC__FL_UTF8_H
#ifdef __cplusplus
extern "C" {
#endif

unsigned short XUtf8IsNonSpacing(unsigned int ucs);

*/
#endif

*/
 Note to editors:
----------------
This file may only contain common, platform-independent function
declarations used internally in FLTK. It may be #included everywhere
in source files in the library, but not in public header files.
*/
#endif
#ifdef __cplusplus
}
#endif

/* Internal UTF-8 header file for the Fast Light Tool Kit (FLTK).
Copyright 1998-2016 by Bill Spitzak and others.
This library is free software. Distribution and use rights are outlined in
the file "COPYING" which should have been included with this file. If this
file is missing or damaged, see the license at:
https://www.fltk.org/COPYING.php
Please see the following page on how to report bugs and issues:
https://www.fltk.org/bugs.php
*/

Generated by Doxygen
13.251 vsnprintf.c File Reference

Portable vsnprintf() implementation.

#include <stdio.h>
#include "flstring.h"

Functions

- int fl_snprintf (char *str, size_t size, const char *fmt,...)
- int fl_vsnprintf (char *buffer, size_t bufsize, const char *format, va_list ap)

FLTKit’s platform independent wrapper for the vsnprintf() C library function.

13.251.1 Detailed Description

Portable vsnprintf() implementation.

13.251.2 Function Documentation

13.251.2.1 fl_vsnprintf()

int fl_vsnprintf (char *buffer, size_t bufsize, const char *format, va_list ap)

FLTKit’s platform independent wrapper for the vsnprintf() C library function.

This function guarantees:

- access to vsnprintf(), even on systems that don’t have it (FLTKit’s own built-in code is used)
- Guarantees NUL termination. Even if string expands larger than the buffer, a terminating NUL is included, unlike some implementations of vsnprintf(), notably Microsoft Visual Studio (pre-2015), which can leave the string unterminated when truncated.

If the build environment for FLTKit has vsnprintf(), fl_vsnprintf() is just a wrapper around the compiler’s provided function. Otherwise, if the function is NOT available, FLTKit’s own built-in version is provided.

The FLTKit built in provides these style options:

- %[ -+#]
- * – padding width
- .* – precision width
- Data types: h, l, ll, L

Generated by Doxygen
• Floating point formats: E, G, e, f, g
• Integer formats: B, X, b, d, i, o, u, x
• Pointer format: p
• String/char: c, s, n

13.252 Xutf8.h

```c
/*
 * Author: Jean-Marc Lienher (http://oksid.ch)
 * Copyright 2000-2010 by O'ksi'D.
 * Please see the following page on how to report bugs and issues:
 * https://www.fltk.org/bugs.php
 */

#ifndef _Xutf8_h
#define _Xutf8_h

#ifdef __cplusplus
extern "C" {
#endif

#include <X11/X.h>
#include <X11/Xlib.h>
#include <X11/Xlocale.h>
#include <X11/Xutil.h>
#include <FL/Fl_Export.H>

typedef struct {
    int nb_font;
    char **font_name_list;
    int *encodings;
    XFontStruct **fonts;
    Font fid;
    int ascent;
    int descent;
    int *ranges;
} XUtf8FontStruct;

XUtf8FontStruct *
XCreateUtf8FontStruct (
    Display *dpy,
    const char *base_font_name_list);

void
XUtf8DrawString(
    Display *display,
    Drawable d,
    XUtf8FontStruct *font_set,
    GC gc,
    int x,
    int y,
    const char *string,
    int num_bytes);

void
XUtf8DrawRtlString(
    Display *display,
    Drawable d,
    XUtf8FontStruct *font_set,
    GC gc,
    int x,
    int y,
    const char *string,
    int num_bytes);

void
XUtf8_measure_extents(
    Display *display,
    Drawable d,
    XUtf8FontStruct *font_set,
    GC gc,
    int *xx,
    int *yy,
    int *ww,
    int *hh,
    const char *string,
    int num_bytes);

void
XUtf8DrawRtlString(
    Display *display,
    Drawable d,
    XUtf8FontStruct *font_set,
```
void XUtf8DrawImageString(Display *display, Drawable d, GC gc, int x, int y, const char *string, int num_bytes);

void XUtf8DrawImageString(Display *display, Drawable d, GC gc, int x, int y, const char *string, int num_bytes);

int XUtf8TextWidth(XUtf8FontStruct *font_set, const char *string, int num_bytes);

int XUtf8UcsWidth(XUtf8FontStruct *font_set, unsigned int ucs);

FL_EXPORT int fl_XGetUtf8FontAndGlyph(XUtf8FontStruct *font_set, unsigned int ucs, XFontStruct **fnt, unsigned short *id);

void XFreeUtf8FontStruct(Display *dpy, XUtf8FontStruct *font_set);

int XConvertUtf8ToUcs(const unsigned char *buf, int len, unsigned int *ucs);

int XConvertUcsToUtf8(unsigned int ucs, char *buf);

int XUtf8CharByteLen(const unsigned char *buf, int len);

int XCountUtf8Char(const unsigned char *buf, int len);

int XFastConvertUtf8ToUcs(const unsigned char *buf, int len, unsigned int *ucs);

long XKeysymToUcs(KeySym keysym);

#define X_HAVE_UTF8_STRING
#define XUtf8LookupString Xutf8LookupString
#define XUtf8LookupString Xutf8LookupString

#endif

#ifdef __cplusplus

#endif

Generated by Doxygen
case.h

```c
/* spacing */
static const unsigned short ucs_table_0041[] = {
    0x0061, /* U+0041 */
    0x0062, /* U+0042 */
    0x0063, /* U+0043 */
    0x0064, /* U+0044 */
    0x0065, /* U+0045 */
    0x0066, /* U+0046 */
    0x0067, /* U+0047 */
    0x0068, /* U+0048 */
    0x0069, /* U+0049 */
    0x006A, /* U+004A */
    0x006B, /* U+004B */
    0x006C, /* U+004C */
    0x006D, /* U+004D */
    0x006E, /* U+004E */
    0x006F, /* U+004F */
    0x0070, /* U+0050 */
    0x0071, /* U+0051 */
    0x0072, /* U+0052 */
    0x0073, /* U+0053 */
    0x0074, /* U+0054 */
    0x0075, /* U+0055 */
    0x0076, /* U+0056 */
    0x0077, /* U+0057 */
    0x0078, /* U+0058 */
    0x0079, /* U+0059 */
    0x007A, /* U+005A */
    0x00,   /* U+0060 */
    0x00,   /* U+0061 */
    0x00,   /* U+0062 */
    0x00,   /* U+0063 */
    0x00,   /* U+0064 */
    0x00,   /* U+0065 */
    0x00,   /* U+0066 */
    0x00,   /* U+0067 */
    0x00,   /* U+0068 */
    0x00,   /* U+0069 */
    0x00,   /* U+006A */
    0x00,   /* U+006B */
    0x00,   /* U+006C */
    0x00,   /* U+006D */
    0x00,   /* U+006E */
    0x00,   /* U+006F */
    0x00,   /* U+0070 */
    0x00,   /* U+0071 */
    0x00,   /* U+0072 */
    0x00,   /* U+0073 */
    0x00,   /* U+0074 */
    0x00,   /* U+0075 */
    0x00,   /* U+0076 */
    0x00,   /* U+0077 */
    0x00,   /* U+0078 */
    0x00,   /* U+0079 */
    0x00,   /* U+007A */
};
```
00081 0x00,
00082 0x00,
00083 0x00,
00084 0x00,
00085 0x00,
00086 0x00,
00087 0x00,
00088 0x00,
00089 0x00,
00090 0x00,
00091 0x00,
00092 0x00,
00093 0x00,
00094 0x00,
00095 0x00,
00096 0x00,
00097 0x00,
00098 0x00,
00099 0x00,
0100 0x00,
0101 0x00,
0102 0x00,
0103 0x00,
0104 0x00,
0105 0x00,
0106 0x00,
0107 0x00,
0108 0x00,
0109 0x00,
0110 0x00,
0111 0x00,
0112 0x00,
0113 0x00,
0114 0x00,
0115 0x00,
0116 0x00,
0117 0x00,
0118 0x00,
0119 0x00,
0120 0x00,
0121 0x00,
0122 0x00,
0123 0x00,
0124 0x00,
0125 0x00,
0126 0x00,
0127 0x00,
0128 0x00,
0129 0x00,
0130 0x00,
0131 /* U+00D0 */ 0x00E0,
0132 /* U+00D1 */ 0x00E1,
0133 /* U+00D2 */ 0x00E2,
0134 /* U+00D3 */ 0x00E3,
0135 /* U+00D4 */ 0x00E4,
0136 /* U+00D5 */ 0x00E5,
0137 /* U+00D6 */ 0x00E6,
0138 /* U+00D7 */ 0x00E7,
0139 /* U+00D8 */ 0x00E8,
0140 /* U+00D9 */ 0x00E9,
0141 /* U+00DA */ 0x00EA,
0142 /* U+00DB */ 0x00EB,
0143 /* U+00DC */ 0x00EC,
0144 /* U+00DD */ 0x00ED,
0145 /* U+00DE */ 0x00EE,
0146 /* U+00DF */ 0x00EF,
0147 /* U+00E0 */ 0x00F0,
0148 /* U+00E1 */ 0x00F1,
0149 /* U+00E2 */ 0x00F2,
0150 /* U+00E3 */ 0x00F3,
0151 /* U+00E4 */ 0x00F4,
0152 /* U+00E5 */ 0x00F5,
0153 /* U+00E6 */ 0x00F6,
0154 0x00,
0155 /* U+00E8 */ 0x00F8,
0156 /* U+00E9 */ 0x00F9,
0157 /* U+00FA */ 0x00FA,
0158 /* U+00FB */ 0x00FC,
0159 /* U+00FD */ 0x00FD,
0160 /* U+00FE */ 0x00FE,
0161 /* U+00FF */ 0x00FF,
0162 0x00,
0163 0x00,
0164 0x00,
0165 0x00,
0166 0x00,
0167 0x00,
0x0342 /* U+0193 */ 0x0260,
0x0343 /* U+0194 */ 0x0263,
0x0344 0x00,
0x0345 /* U+0196 */ 0x0269,
0x0346 /* U+0197 */ 0x0268,
0x0347 /* U+0198 */ 0x0199,
0x0348 0x00,
0x0349 0x00,
0x0350 0x00,
0x0351 /* U+019C */ 0x026F,
0x0352 /* U+019D */ 0x0272,
0x0353 0x00,
0x0354 /* U+019F */ 0x00,
0x0355 /* U+01A0 */ 0x0199,
0x0356 0x00,
0x0357 /* U+01A2 */ 0x01A3,
0x0358 0x00,
0x0359 /* U+01A4 */ 0x01A5,
0x035A 0x00,
0x035B /* U+01A9 */ 0x0283,
0x035C 0x00,
0x035D 0x00,
0x035E /* U+01AC */ 0x01AD,
0x035F 0x00,
0x0360 /* U+01AE */ 0x01A1,
0x0361 0x00,
0x0362 /* U+01A7 */ 0x01A8,
0x0363 0x00,
0x0364 /* U+01A9 */ 0x0283,
0x0365 0x00,
0x0366 0x00,
0x0367 /* U+01AC */ 0x01AD,
0x0368 0x00,
0x0369 /* U+01AE */ 0x0288,
0x0370 /* U+01AF */ 0x01B0,
0x0371 0x00,
0x0372 /* U+01B1 */ 0x02BA,
0x0373 /* U+01B2 */ 0x02BB,
0x0374 /* U+01B3 */ 0x01B4,
0x0375 0x00,
0x0376 /* U+01B5 */ 0x01B6,
0x0377 0x00,
0x0378 /* U+01B7 */ 0x0292,
0x0379 /* U+01B8 */ 0x01B9,
0x0380 0x00,
0x0381 0x00,
0x0382 0x00,
0x0383 /* U+01BD */ 0x01BD,
0x0384 0x00,
0x0385 0x00,
0x0386 0x00,
0x0387 0x00,
0x0388 0x00,
0x0389 0x00,
0x0390 0x00,
0x0391 /* U+01C4 */ 0x01C6,
0x0392 /* U+01C5 */ 0x01C9,
0x0393 0x00,
0x0394 /* U+01C7 */ 0x01C8,
0x0395 /* U+01CA */ 0x01CC,
0x0396 0x00,
0x0397 /* U+01CB */ 0x01CH,
0x0398 0x00,
0x0399 0x00,
0x0400 /* U+01CD */ 0x01CE,
0x0401 0x00,
0x0402 /* U+01CF */ 0x01D0,
0x0403 0x00,
0x0404 /* U+01D0 */ 0x01D2,
0x0405 0x00,
0x0406 /* U+01D1 */ 0x01D4,
0x0407 0x00,
0x0408 /* U+01D5 */ 0x01D6,
0x0409 0x00,
0x040A /* U+01D7 */ 0x01DA,
0x040B 0x00,
0x040C /* U+01D8 */ 0x01DB,
0x040D 0x00,
0x040E /* U+01D9 */ 0x01DC,
0x040F 0x00,
0x0410 /* U+01DE */ 0x01DF,
0x0411 0x00,
0x0412 /* U+01E0 */ 0x01E1,
0x0413 0x00,
0x0414 /* U+01E2 */ 0x01E3,
0x0415 0x00,
0x0416 /* U+01E4 */ 0x01E5,
0x0417 0x00,
0x0418 /* U+01E6 */ 0x01E7,
0x0419 0x00,
00429 /* U+01EA */ 0x01EB,  
00430 0x00,  
00431 /* U+01EC */ 0x01ED,  
00432 0x00,  
00433 /* U+01EE */ 0x01EF,  
00434 0x00,  
00435 0x00,  
00436 /* U+01F1 */ 0x01F3,  
00437 /* U+01F2 */ 0x00,  
00438 0x00,  
00439 /* U+01F4 */ 0x01F5,  
00440 0x00,  
00441 0x00,  
00442 0x00,  
00443 0x00,  
00444 0x00,  
00445 /* U+01FA */ 0x01FB,  
00446 0x00,  
00447 /* U+01FC */ 0x01FD,  
00448 0x00,  
00449 /* U+01FE */ 0x01FF,  
00450 0x00,  
00451 /* U+0200 */ 0x0201,  
00452 0x00,  
00453 /* U+0202 */ 0x0203,  
00454 0x00,  
00455 /* U+0204 */ 0x0205,  
00456 0x00,  
00457 /* U+0206 */ 0x0207,  
00458 0x00,  
00459 /* U+0208 */ 0x0209,  
00460 0x00,  
00461 /* U+020A */ 0x020B,  
00462 0x00,  
00463 /* U+020C */ 0x020D,  
00464 0x00,  
00465 /* U+020E */ 0x020F,  
00466 0x00,  
00467 /* U+0210 */ 0x0211,  
00468 0x00,  
00469 /* U+0212 */ 0x0213,  
00470 0x00,  
00471 /* U+0214 */ 0x0215,  
00472 0x00,  
00473 /* U+0216 */ 0x0217,  
00474 0x00,  
00475 0x00,  
00476 0x00,  
00477 0x00,  
00478 0x00,  
00479 0x00,  
00480 0x00,  
00481 0x00,  
00482 0x00,  
00483 0x00,  
00484 0x00,  
00485 0x00,  
00486 0x00,  
00487 0x00,  
00488 0x00,  
00489 0x00,  
00490 0x00,  
00491 0x00,  
00492 0x00,  
00493 0x00,  
00494 0x00,  
00495 0x00,  
00496 0x00,  
00497 0x00,  
00498 0x00,  
00499 0x00,  
0500 0x00,  
0501 0x00,  
0502 0x00,  
0503 0x00,  
0504 0x00,  
0505 0x00,  
0506 0x00,  
0507 0x00,  
0508 0x00,  
0509 0x00,  
0510 0x00,  
0511 0x00,  
0512 0x00,  
0513 0x00,  
0514 0x00,  
0515 0x00,
00516 0x00,
00517 0x00,
00518 0x00,
00519 0x00,
00520 0x00,
00521 0x00,
00522 0x00,
00523 0x00,
00524 0x00,
00525 0x00,
00526 0x00,
00527 0x00,
00528 0x00,
00529 0x00,
00530 0x00,
00531 0x00,
00532 0x00,
00533 0x00,
00534 0x00,
00535 0x00,
00536 0x00,
00537 0x00,
00538 0x00,
00539 0x00,
00540 0x00,
00541 0x00,
00542 0x00,
00543 0x00,
00544 0x00,
00545 0x00,
00546 0x00,
00547 0x00,
00548 0x00,
00549 /* U+0262 */ 0x0,
00550 0x00,
00551 0x00,
00552 0x00,
00553 0x00,
00554 0x00,
00555 0x00,
00556 0x00,
00557 /* U+026A */ 0x0,
00558 0x00,
00559 0x00,
00560 0x00,
00561 0x00,
00562 0x00,
00563 0x00,
00564 0x00,
00565 0x00,
00566 0x00,
00567 /* U+0274 */ 0x0,
00568 0x00,
00569 /* U+0276 */ 0x0,
00570 0x00,
00571 0x00,
00572 0x00,
00573 0x00,
00574 0x00,
00575 0x00,
00576 0x00,
00577 0x00,
00578 0x00,
00579 /* U+0280 */ 0x0,
00580 /* U+0281 */ 0x0,
00581 0x00,
00582 0x00,
00583 0x00,
00584 0x00,
00585 0x00,
00586 0x00,
00587 0x00,
00588 0x00,
00589 0x00,
00590 0x00,
00591 0x00,
00592 0x00,
00593 0x00,
00594 /* U+028F */ 0x0,
00595 0x00,
00596 0x00,
00597 0x00,
00598 0x00,
00599 0x00,
00600 0x00,
00601 0x00,
00602 0x00,
00603 0x00,
00604 0x00,
00605 0x00,
00606 0x00,
00607 0x00,
00608 0x00,
00609 0x00,
00610 0x00,
00611 0x00,
00612 0x00,
00613 0x00,
00614 0x00,
00615 0x00,
00616 0x00,
00617 0x00,
00618 0x00,
00619 0x00,
00620 0x00,
00621 0x00,
00622 0x00,
00623 0x00,
00624 0x00,
00625 0x00,
00626 0x00,
00627 0x00,
00628 0x00,
00629 0x00,
00630 0x00,
00631 0x00,
00632 0x00,
00633 0x00,
00634 0x00,
00635 0x00,
00636 0x00,
00637 0x00,
00951 0x00,
00952 /* U+04C1 */ 0x00,
00953 0x00,
00954 /* U+04C3 */ 0x00,
00955 0x00,
00956 0x00,
00957 0x00,
00958 /* U+04C7 */ 0x00,
00959 0x00,
00960 0x00,
00961 0x00,
00962 /* U+04CB */ 0x00,
00963 0x00,
00964 0x00,
00965 0x00,
00966 0x00,
00967 /* U+04D0 */ 0x00,
00968 0x00,
00969 /* U+04D2 */ 0x00,
00970 0x00,
00971 /* U+04D4 */ 0x00,
00972 0x00,
00973 /* U+04D6 */ 0x00,
00974 0x00,
00975 /* U+04D8 */ 0x00,
00976 0x00,
00977 /* U+04DA */ 0x00,
00978 0x00,
00979 /* U+04DC */ 0x00,
00980 0x00,
00981 /* U+04DE */ 0x00,
00982 0x00,
00983 /* U+04EE */ 0x00,
00984 0x00,
00985 /* U+04F0 */ 0x00,
00986 0x00,
00987 /* U+04F2 */ 0x00,
00988 0x00,
00989 /* U+04F4 */ 0x00,
00990 0x00,
00991 /* U+04F6 */ 0x00,
00992 0x00,
00993 /* U+04F8 */ 0x00,
00994 0x00,
00995 0x00,
00996 0x00,
00997 /* U+04F0 */ 0x00,
00998 0x00,
00999 /* U+04F2 */ 0x00,
01000 0x00,
01001 /* U+04F4 */ 0x00,
01002 0x00,
01003 /* U+04F6 */ 0x00,
01004 0x00,
01005 0x00,
01006 0x00,
01007 /* U+04F8 */ 0x00,
01008 0x00,
01009 0x00,
static const unsigned short ucs_table_10A0[] = {
    /* U+10A0 */ 0x10D0,
    /* U+10A1 */ 0x10D1,
    /* U+10A2 */ 0x10D2,
    /* U+10A3 */ 0x10D3,
    /* U+10A4 */ 0x10D4,
    /* U+10A5 */ 0x10D5,
    /* U+10A6 */ 0x10D6,
    /* U+10A7 */ 0x10D7,
    /* U+10A8 */ 0x10D8,
    /* U+10A9 */ 0x10D9,
    /* U+10AA */ 0x10DA,
    /* U+10AB */ 0x10DB,
    /* U+10AC */ 0x10DC,
    /* U+10AD */ 0x10DD,
    /* U+10AE */ 0x10DE,
    /* U+10AF */ 0x10DF,
    /* U+10B0 */ 0x10E0,
    /* U+10B1 */ 0x10E1,
    /* U+10B2 */ 0x10E2,
01125 / * U+10B4 */ 0x10E4,
01126 / * U+10B5 */ 0x10E5,
01127 / * U+10B6 */ 0x10E6,
01128 / * U+10B7 */ 0x10E7,
01129 / * U+10B8 */ 0x10E8,
01130 / * U+10B9 */ 0x10E9,
01131 / * U+10BA */ 0x10EA,
01132 / * U+10BB */ 0x10EB,
01133 / * U+10BC */ 0x10EC,
01134 / * U+10BD */ 0x10ED,
01135 / * U+10BE */ 0x10EE,
01136 / * U+10BF */ 0x10EF,
01137 / * U+10C0 */ 0x10F0,
01138 / * U+10C1 */ 0x10F1,
01139 / * U+10C2 */ 0x10F2,
01140 / * U+10C3 */ 0x10F3,
01141 / * U+10C4 */ 0x10F4,
01142 / * U+10C5 */ 0x10F5,
01143 */
01144 01145 static const unsigned short ucs_table_1E00[] = {
01146 01147 0x00,
01148 01149 0x00,
01150 01151 0x00,
01152 01153 0x00,
01154 01155 0x00,
01156 01157 0x00,
01158 01159 0x00,
01160 01161 0x00,
01162 01163 0x00,
01164 01165 0x00,
01166 01167 0x00,
01168 01169 0x00,
01170 01171 0x00,
01172 01173 0x00,
01174 01175 0x00,
01176 01177 0x00,
01178 01179 0x00,
01180 01181 0x00,
01182 01183 0x00,
01184 01185 0x00,
01186 01187 0x00,
01188 01189 0x00,
01190 01191 0x00,
01192 01193 0x00,
01194 01195 0x00,
01196 01197 0x00,
01198 01199 0x00,
01200 01201 0x00,
01202 01203 0x00,
01204 01205 0x00,
01206 01207 0x00,
01208 01209 0x00,
01210 01211 0x00,
01212 /* \+1E42 */ 0x1E43,
01213 0x00,
01214 /* \+1E44 */ 0x1E45,
01215 0x00,
01216 /* \+1E46 */ 0x1E47,
01217 0x00,
01218 /* \+1E48 */ 0x1E49,
01219 0x00,
01220 /* \+1E4A */ 0x1E4B,
01221 0x00,
01222 /* \+1E4C */ 0x1E4D,
01223 0x00,
01224 /* \+1E4E */ 0x1E4F,
01225 0x00,
01226 /* \+1E50 */ 0x1E51,
01227 0x00,
01228 /* \+1E52 */ 0x1E53,
01229 0x00,
01230 /* \+1E54 */ 0x1E55,
01231 0x00,
01232 /* \+1E56 */ 0x1E57,
01233 0x00,
01234 /* \+1E58 */ 0x1E59,
01235 0x00,
01236 /* \+1E5A */ 0x1E5B,
01237 0x00,
01238 /* \+1E5C */ 0x1E5D,
01239 0x00,
01240 /* \+1E5E */ 0x1E5F,
01241 0x00,
01242 /* \+1E60 */ 0x1E61,
01243 0x00,
01244 /* \+1E62 */ 0x1E63,
01245 0x00,
01246 /* \+1E64 */ 0x1E65,
01247 0x00,
01248 /* \+1E66 */ 0x1E67,
01249 0x00,
01250 /* \+1E68 */ 0x1E69,
01251 0x00,
01252 /* \+1E6A */ 0x1E6B,
01253 0x00,
01254 /* \+1E6C */ 0x1E6D,
01255 0x00,
01256 /* \+1E6E */ 0x1E6F,
01257 0x00,
01258 /* \+1E70 */ 0x1E71,
01259 0x00,
01260 /* \+1E72 */ 0x1E73,
01261 0x00,
01262 /* \+1E74 */ 0x1E75,
01263 0x00,
01264 /* \+1E76 */ 0x1E77,
01265 0x00,
01266 /* \+1E78 */ 0x1E79,
01267 0x00,
01268 /* \+1E7A */ 0x1E7B,
01269 0x00,
01270 /* \+1E7C */ 0x1E7D,
01271 0x00,
01272 /* \+1E7E */ 0x1E7F,
01273 0x00,
01274 /* \+1E80 */ 0x1E81,
01275 0x00,
01276 /* \+1E82 */ 0x1E83,
01277 0x00,
01278 /* \+1E84 */ 0x1E85,
01279 0x00,
01280 /* \+1E86 */ 0x1E87,
01281 0x00,
01282 /* \+1E88 */ 0x1E89,
01283 0x00,
01284 /* \+1E8A */ 0x1E8B,
01285 0x00,
01286 /* \+1E8C */ 0x1E8D,
01287 0x00,
01288 /* \+1E8E */ 0x1E8F,
01289 0x00,
01290 /* \+1E90 */ 0x1E91,
01291 0x00,
01292 /* \+1E92 */ 0x1E93,
01293 0x00,
01294 /* \+1E94 */ 0x1E95,
<table>
<thead>
<tr>
<th>Line</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>01299</td>
<td>0x00,</td>
</tr>
<tr>
<td>01300</td>
<td>0x00,</td>
</tr>
<tr>
<td>01301</td>
<td>0x00,</td>
</tr>
<tr>
<td>01302</td>
<td>0x00,</td>
</tr>
<tr>
<td>01303</td>
<td>0x00,</td>
</tr>
<tr>
<td>01304</td>
<td>0x00,</td>
</tr>
<tr>
<td>01305</td>
<td>0x00,</td>
</tr>
<tr>
<td>01306</td>
<td>/* U+1EA0 */ 0x1EA1,</td>
</tr>
<tr>
<td>01307</td>
<td>0x00,</td>
</tr>
<tr>
<td>01308</td>
<td>/* U+1EA2 */ 0x1EA3,</td>
</tr>
<tr>
<td>01309</td>
<td>0x00,</td>
</tr>
<tr>
<td>01310</td>
<td>/* U+1EA4 */ 0x1EA5,</td>
</tr>
<tr>
<td>01311</td>
<td>0x00,</td>
</tr>
<tr>
<td>01312</td>
<td>/* U+1EA6 */ 0x1EA7,</td>
</tr>
<tr>
<td>01313</td>
<td>0x00,</td>
</tr>
<tr>
<td>01314</td>
<td>/* U+1EA8 */ 0x1EA9,</td>
</tr>
<tr>
<td>01315</td>
<td>0x00,</td>
</tr>
<tr>
<td>01316</td>
<td>/* U+1EAA */ 0x1EAB,</td>
</tr>
<tr>
<td>01317</td>
<td>0x00,</td>
</tr>
<tr>
<td>01318</td>
<td>/* U+1EAC */ 0x1EAD,</td>
</tr>
<tr>
<td>01319</td>
<td>0x00,</td>
</tr>
<tr>
<td>01320</td>
<td>/* U+1EAE */ 0x1EAF,</td>
</tr>
<tr>
<td>01321</td>
<td>0x00,</td>
</tr>
<tr>
<td>01322</td>
<td>/* U+1EB0 */ 0x1EB1,</td>
</tr>
<tr>
<td>01323</td>
<td>0x00,</td>
</tr>
<tr>
<td>01324</td>
<td>/* U+1EB2 */ 0x1EB3,</td>
</tr>
<tr>
<td>01325</td>
<td>0x00,</td>
</tr>
<tr>
<td>01326</td>
<td>/* U+1EB4 */ 0x1EB5,</td>
</tr>
<tr>
<td>01327</td>
<td>0x00,</td>
</tr>
<tr>
<td>01328</td>
<td>/* U+1EB6 */ 0x1EB7,</td>
</tr>
<tr>
<td>01329</td>
<td>0x00,</td>
</tr>
<tr>
<td>01330</td>
<td>/* U+1EB8 */ 0x1EB9,</td>
</tr>
<tr>
<td>01331</td>
<td>0x00,</td>
</tr>
<tr>
<td>01332</td>
<td>/* U+1EBA */ 0x1EBA,</td>
</tr>
<tr>
<td>01333</td>
<td>0x00,</td>
</tr>
<tr>
<td>01334</td>
<td>/* U+1EBC */ 0x1EBD,</td>
</tr>
<tr>
<td>01335</td>
<td>0x00,</td>
</tr>
<tr>
<td>01336</td>
<td>/* U+1EBE */ 0x1EBF,</td>
</tr>
<tr>
<td>01337</td>
<td>0x00,</td>
</tr>
<tr>
<td>01338</td>
<td>/* U+1ECE */ 0x1EC1,</td>
</tr>
<tr>
<td>01339</td>
<td>0x00,</td>
</tr>
<tr>
<td>01340</td>
<td>/* U+1EC2 */ 0x1EC3,</td>
</tr>
<tr>
<td>01341</td>
<td>0x00,</td>
</tr>
<tr>
<td>01342</td>
<td>/* U+1EC4 */ 0x1EC5,</td>
</tr>
<tr>
<td>01343</td>
<td>0x00,</td>
</tr>
<tr>
<td>01344</td>
<td>/* U+1EC6 */ 0x1EC7,</td>
</tr>
<tr>
<td>01345</td>
<td>0x00,</td>
</tr>
<tr>
<td>01346</td>
<td>/* U+1EC8 */ 0x1EC9,</td>
</tr>
<tr>
<td>01347</td>
<td>0x00,</td>
</tr>
<tr>
<td>01348</td>
<td>/* U+1ECA */ 0x1ECB,</td>
</tr>
<tr>
<td>01349</td>
<td>0x00,</td>
</tr>
<tr>
<td>01350</td>
<td>/* U+1ECC */ 0x1ECD,</td>
</tr>
<tr>
<td>01351</td>
<td>0x00,</td>
</tr>
<tr>
<td>01352</td>
<td>/* U+1ECE */ 0x1ECF,</td>
</tr>
<tr>
<td>01353</td>
<td>0x00,</td>
</tr>
<tr>
<td>01354</td>
<td>/* U+1ED0 */ 0x1ED1,</td>
</tr>
<tr>
<td>01355</td>
<td>0x00,</td>
</tr>
<tr>
<td>01356</td>
<td>/* U+1ED2 */ 0x1ED3,</td>
</tr>
<tr>
<td>01357</td>
<td>0x00,</td>
</tr>
<tr>
<td>01358</td>
<td>/* U+1ED4 */ 0x1ED5,</td>
</tr>
<tr>
<td>01359</td>
<td>0x00,</td>
</tr>
<tr>
<td>01360</td>
<td>/* U+1ED6 */ 0x1ED7,</td>
</tr>
<tr>
<td>01361</td>
<td>0x00,</td>
</tr>
<tr>
<td>01362</td>
<td>/* U+1ED8 */ 0x1ED9,</td>
</tr>
<tr>
<td>01363</td>
<td>0x00,</td>
</tr>
<tr>
<td>01364</td>
<td>/* U+1EDA */ 0x1EE0,</td>
</tr>
<tr>
<td>01365</td>
<td>0x00,</td>
</tr>
<tr>
<td>01366</td>
<td>/* U+1EDC */ 0x1EDE,</td>
</tr>
<tr>
<td>01367</td>
<td>0x00,</td>
</tr>
<tr>
<td>01368</td>
<td>/* U+1EDE */ 0x1EFD,</td>
</tr>
<tr>
<td>01369</td>
<td>0x00,</td>
</tr>
<tr>
<td>01370</td>
<td>/* U+1EE0 */ 0x1EE1,</td>
</tr>
<tr>
<td>01371</td>
<td>0x00,</td>
</tr>
<tr>
<td>01372</td>
<td>/* U+1EE2 */ 0x1EE3,</td>
</tr>
<tr>
<td>01373</td>
<td>0x00,</td>
</tr>
<tr>
<td>01374</td>
<td>/* U+1EE4 */ 0x1EE5,</td>
</tr>
<tr>
<td>01375</td>
<td>0x00,</td>
</tr>
<tr>
<td>01376</td>
<td>/* U+1EE6 */ 0x1EE7,</td>
</tr>
<tr>
<td>01377</td>
<td>0x00,</td>
</tr>
<tr>
<td>01378</td>
<td>/* U+1EE8 */ 0x1EE9,</td>
</tr>
<tr>
<td>01379</td>
<td>0x00,</td>
</tr>
<tr>
<td>01380</td>
<td>/* U+1EEA */ 0x1EEB,</td>
</tr>
<tr>
<td>01381</td>
<td>0x00,</td>
</tr>
<tr>
<td>01382</td>
<td>/* U+1EED */ 0x1EEF,</td>
</tr>
<tr>
<td>01383</td>
<td>0x00,</td>
</tr>
<tr>
<td>01384</td>
<td>/* U+1EEE */ 0x1EEF,</td>
</tr>
<tr>
<td>01385</td>
<td>0x00,</td>
</tr>
</tbody>
</table>

Generated by Doxygen
01386 /* U+1EF0 */ 0x1EF1,
01387 0x00,
01388 /* U+1EF2 */ 0x1EF3,
01389 0x00,
01390 /* U+1EF4 */ 0x1EF5,
01391 0x00,
01392 /* U+1EF6 */ 0x1EF7,
01393 0x00,
01394 /* U+1EF8 */ 0x1EF9,
01395 0x00,
01396 0x00,
01397 0x00,
01398 0x00,
01399 0x00,
01400 0x00,
01401 0x00,
01402 0x00,
01403 0x00,
01404 0x00,
01405 0x00,
01406 0x00,
01407 0x00,
01408 0x00,
01409 0x00,
01410 /* U+1F08 */ 0x1F00,
01411 /* U+1F09 */ 0x1F01,
01412 /* U+1F0A */ 0x1F02,
01413 /* U+1F0B */ 0x1F03,
01414 /* U+1F0C */ 0x1F04,
01415 /* U+1F0D */ 0x1F05,
01416 /* U+1F0E */ 0x1F06,
01417 /* U+1F0F */ 0x1F07,
01418 0x00,
01419 0x00,
01420 0x00,
01421 0x00,
01422 0x00,
01423 0x00,
01424 0x00,
01425 0x00,
01426 /* U+1F18 */ 0x1F10,
01427 /* U+1F19 */ 0x1F11,
01428 /* U+1F1A */ 0x1F12,
01429 /* U+1F1B */ 0x1F13,
01430 /* U+1F1C */ 0x1F14,
01431 /* U+1F1D */ 0x1F15,
01432 0x00,
01433 0x00,
01434 0x00,
01435 0x00,
01436 0x00,
01437 0x00,
01438 0x00,
01439 0x00,
01440 0x00,
01441 0x00,
01442 /* U+1F28 */ 0x1F20,
01443 /* U+1F29 */ 0x1F21,
01444 /* U+1F2A */ 0x1F22,
01445 /* U+1F2B */ 0x1F23,
01446 /* U+1F2C */ 0x1F24,
01447 /* U+1F2D */ 0x1F25,
01448 /* U+1F2E */ 0x1F26,
01449 /* U+1F2F */ 0x1F27,
01450 0x00,
01451 0x00,
01452 0x00,
01453 0x00,
01454 0x00,
01455 0x00,
01456 0x00,
01457 0x00,
01458 /* U+1F38 */ 0x1F30,
01459 /* U+1F39 */ 0x1F31,
01460 /* U+1F3A */ 0x1F32,
01461 /* U+1F3B */ 0x1F33,
01462 /* U+1F3C */ 0x1F34,
01463 /* U+1F3D */ 0x1F35,
01464 /* U+1F3E */ 0x1F36,
01465 /* U+1F3F */ 0x1F37,
01466 0x00,
01467 0x00,
01468 0x00,
01469 0x00,
01470 0x00,
01471 0x00,
01472 0x00,
13.253 case.h

01473 0x00,
01474 /* U+1F48 */ 0x1F40,
01475 /* U+1F49 */ 0x1F41,
01476 /* U+1F4A */ 0x1F42,
01477 /* U+1F4B */ 0x1F43,
01478 /* U+1F4C */ 0x1F44,
01479 /* U+1F4D */ 0x1F45,
01480 0x00,
01484 0x00,
01485 0x00,
01486 0x00,
01487 0x00,
01488 0x00,
01489 0x00,
01490 0x00,
01491 /* U+1F59 */ 0x1F51,
01492 0x00,
01493 /* U+1F5B */ 0x1F53,
01494 0x00,
01495 /* U+1F5D */ 0x1F55,
01496 0x00,
01497 /* U+1F5F */ 0x1F57,
01498 0x00,
01499 0x00,
01500 0x00,
01501 0x00,
01502 0x00,
01503 0x00,
01504 0x00,
01505 0x00,
01506 /* U+1F68 */ 0x1F60,
01507 /* U+1F69 */ 0x1F61,
01508 /* U+1F6A */ 0x1F62,
01509 /* U+1F6B */ 0x1F63,
01510 /* U+1F6C */ 0x1F64,
01511 /* U+1F6D */ 0x1F65,
01512 /* U+1F6E */ 0x1F66,
01513 /* U+1F6F */ 0x1F67,
01514 0x00,
01515 0x00,
01516 0x00,
01517 0x00,
01518 0x00,
01519 0x00,
01520 0x00,
01521 0x00,
01522 0x00,
01523 0x00,
01524 0x00,
01525 0x00,
01526 0x00,
01527 0x00,
01528 0x00,
01529 0x00,
01530 0x00,
01531 0x00,
01532 0x00,
01533 0x00,
01534 0x00,
01535 0x00,
01536 0x00,
01537 0x00,
01538 /* U+1F88 */ 0x0,
01539 /* U+1F89 */ 0x0,
01540 /* U+1F8A */ 0x0,
01541 /* U+1F8B */ 0x0,
01542 /* U+1F8C */ 0x0,
01543 /* U+1F8D */ 0x0,
01544 /* U+1F8E */ 0x0,
01545 /* U+1F8F */ 0x0,
01546 0x00,
01547 0x00,
01548 0x00,
01549 0x00,
01550 0x00,
01551 0x00,
01552 0x00,
01553 0x00,
01554 /* U+1F98 */ 0x0,
01555 /* U+1F99 */ 0x0,
01556 /* U+1F9A */ 0x0,
01557 /* U+1F9B */ 0x0,
01558 /* U+1F9C */ 0x0,
01559 /* U+1F9D */ 0x0,
01560 /* U+1F5E */ 0x0,
01561 /* U+1F5F */ 0x0,
01562 0x00,
01563 0x00,
01564 0x00,
01565 0x00,
01566 0x00,
01567 0x00,
01568 0x00,
01569 0x00,
01570 /* U+1FA8 */ 0x0,
01571 /* U+1FA9 */ 0x0,
01572 /* U+1FAA */ 0x0,
01573 /* U+1FAB */ 0x0,
01574 /* U+1FAC */ 0x0,
01575 /* U+1FAD */ 0x0,
01576 /* U+1FAE */ 0x0,
01577 /* U+1FAF */ 0x0,
01578 0x00,
01579 0x00,
01580 0x00,
01581 0x00,
01582 0x00,
01583 0x00,
01584 0x00,
01585 0x00,
01586 /* U+1FB8 */ 0x0,
01587 0x00,
01588 0x00,
01589 0x00,
01590 0x00,
01591 0x00,
01592 0x00,
01593 0x00,
01594 0x00,
01595 0x00,
01596 0x00,
01597 0x00,
01598 0x00,
01599 0x00,
01600 0x00,
01601 /* U+1FC8 */ 0x0,
01602 /* U+1FC9 */ 0x0,
01603 /* U+1FCA */ 0x0,
01604 /* U+1FCC */ 0x0,
01605 /* U+1FD8 */ 0x0,
01606 /* U+1FD9 */ 0x0,
01607 0x00,
01608 0x00,
01609 0x00,
01610 0x00,
01611 0x00,
01612 0x00,
01613 0x00,
01614 0x00,
01615 0x00,
01616 0x00,
01617 0x00,
01618 /* U+1FDE */ 0x0,
01619 /* U+1FDF */ 0x0,
01620 /* U+1FE0 */ 0x0,
01621 /* U+1FE1 */ 0x0,
01622 0x00,
01623 0x00,
01624 0x00,
01625 0x00,
01626 0x00,
01627 0x00,
01628 0x00,
01629 0x00,
01630 0x00,
01631 0x00,
01632 0x00,
01633 0x00,
01634 /* U+1FE8 */ 0x0,
01635 /* U+1FE9 */ 0x0,
01636 /* U+1FEA */ 0x0,
01637 /* U+1FEB */ 0x0,
01638 /* U+1FEC */ 0x0,
01639 0x00,
13.253 case.h

01647 0x00,
01648 0x00,
01649 0x00,
01650 /* U+1FF8 */ 0x1F78,
01651 /* U+1FF9 */ 0x1F79,
01652 /* U+1FFA */ 0x1F7A,
01653 /* U+1FFB */ 0x1F7B,
01654 /* U+1FFC */ 0x0,
01655 }
01656
01657 static const unsigned short ucs_table_2102[] = {
01658 /* U+2102 */ 0x0,
01659 0x00,
01660 0x00,
01661 0x00,
01662 0x00,
01663 0x00,
01664 0x00,
01665 0x00,
01666 0x00,
01667 /* U+2115 */ 0x00,
01668 0x00,
01669 /* U+2118 */ 0x00,
01670 0x00,
01671 0x00,
01672 /* U+2119 */ 0x00,
01673 /* U+211A */ 0x00,
01674 /* U+211B */ 0x00,
01675 /* U+211E */ 0x00,
01676 /* U+211F */ 0x00,
01677 /* U+2124 */ 0x00,
01678 0x00,
01679 0x00,
01680 /* U+2128 */ 0x00,
01681 /* U+212A */ 0x00,
01682 /* U+212C */ 0x00,
01683 /* U+212E */ 0x00,
01684 /* U+212F */ 0x00,
01685 /* U+2130 */ 0x00,
01686 /* U+2131 */ 0x00,
01687 /* U+2132 */ 0x00,
01688 /* U+2133 */ 0x00,
01689 0x00,
01690 0x00,
01691 0x00,
01692 /* U+2224 */ 0x00,
01693 /* U+2226 */ 0x00,
01694 /* U+2228 */ 0x00,
01695 0x00,
01696 /* U+222A */ 0x00,
01697 /* U+222C */ 0x00,
01698 0x00,
01699 0x00,
01700 /* U+222F */ 0x00,
01701 /* U+2231 */ 0x00,
01702 /* U+2232 */ 0x00,
01703 /* U+2234 */ 0x00,
01704 /* U+2236 */ 0x00,
01705 /* U+2238 */ 0x00,
01706 /* U+223A */ 0x00,
01707 /* U+223C */ 0x00,
01708 }
01709
01710 static const unsigned short ucs_table_24B6[] = {
01711 /* U+24B6 */ 0x020D,
01712 /* U+24B7 */ 0x020D,
01713 /* U+24B8 */ 0x020D,
01714 /* U+24B9 */ 0x020D,
01715 /* U+24BA */ 0x020D,
01716 /* U+24BB */ 0x020D,
01717 /* U+24BC */ 0x020D,
01718 /* U+24BD */ 0x020D,
01719 /* U+24BE */ 0x020D,
01720 /* U+24BF */ 0x020D,
01721 /* U+24C0 */ 0x020D,
01722 /* U+24C1 */ 0x020D,
01723 /* U+24C2 */ 0x020D,
01724 /* U+24C3 */ 0x020D,
01725 /* U+24C4 */ 0x020D,
01726 /* U+24C5 */ 0x020D,
01727 /* U+24C6 */ 0x020D,
01728 /* U+24C7 */ 0x020D,
01729 /* U+24C8 */ 0x020D,
01730 /* U+24C9 */ 0x020D,
01731 /* U+24CA */ 0x020D,
01732 /* U+24CB */ 0x020D,
01733 /* U+24CC */ 0x020D,
01734 /* U+24CD */ 0x020D,
```c
/* U+24CD */ 0x24E7,
/* U+24CE */ 0x24E8,
/* U+24CF */ 0x24E9,
};

static const unsigned short ucs_table_33CE[] = {
/* U+33CE */ 0x0,
};

static const unsigned short ucs_table_FF21[] = {
/* U+FF21 */ 0xFF41,
/* U+FF22 */ 0xFF42,
/* U+FF23 */ 0xFF43,
/* U+FF24 */ 0xFF44,
/* U+FF25 */ 0xFF45,
/* U+FF26 */ 0xFF46,
/* U+FF27 */ 0xFF47,
/* U+FF28 */ 0xFF48,
/* U+FF29 */ 0xFF49,
/* U+FF2A */ 0xFF4A,
/* U+FF2B */ 0xFF4B,
/* U+FF2C */ 0xFF4C,
/* U+FF2D */ 0xFF4D,
/* U+FF2E */ 0xFF4E,
/* U+FF2F */ 0xFF4F,
/* U+FF30 */ 0xFF50,
/* U+FF31 */ 0xFF51,
/* U+FF32 */ 0xFF52,
/* U+FF33 */ 0xFF53,
/* U+FF34 */ 0xFF54,
/* U+FF35 */ 0xFF55,
/* U+FF36 */ 0xFF56,
/* U+FF37 */ 0xFF57,
/* U+FF38 */ 0xFF58,
/* U+FF39 */ 0xFF59,
/* U+FF3A */ 0xFF5A,
};

13.254 dingbats_.h

/* dingbats */

static const char unicode_to_dingbats_1b_0020[] = {
/* U+0020 */ 0x20,
};
```
00046 0x00,
00047 0x00,
00048 0x00,
00049 0x00,
00050 0x00,
00051 0x00,
00052 0x00,
00053 0x00,
00054 0x00,
00055 0x00,
00056 0x00,
00057 0x00,
00058 0x00,
00059 0x00,
00060 0x00,
00061 0x00,
00062 0x00,
00063 0x00,
00064 0x00,
00065 0x00,
00066 0x00,
00067 0x00,
00068 0x00,
00069 0x00,
00070 0x00,
00071 0x00,
00072 0x00,
00073 0x00,
00074 0x00,
00075 0x00,
00076 0x00,
00077 0x00,
00078 0x00,
00079 0x00,
00080 0x00,
00081 0x00,
00082 0x00,
00083 0x00,
00084 0x00,
00085 0x00,
00086 0x00,
00087 0x00,
00088 0x00,
00089 0x00,
00090 0x00,
00091 0x00,
00092 0x00,
00093 0x00,
00094 0x00,
00095 0x00,
00096 0x00,
00097 0x00,
00098 0x00,
00099 0x00,
00100 0x00,
00101 0x00,
00102 0x00,
00103 0x00,
00104 0x00,
00105 0x00,
00106 0x00,
00107 0x00,
00108 0x00,
00109 0x00,
00110 0x00,
00111 0x00,
00112 0x00,
00113 0x00,
00114 0x00,
00115 0x00,
00116 0x00,
00117 0x00,
00118 0x00,
00119 0x00,
00120 0x00,
00121 0x00,
00122 0x00,
00123 0x00,
00124 0x00,
00125 0x00,
00126 0x00,
00127 0x00,
00128 0x00,
00129 0x00,
00130 0x00,
00131 0x00,
00132 /* U+00A0 */ 0x20,
static const char unicode_to_dingbats_1b_2192[] = {
/* U+2192 */ (char)0xD5,

static const char unicode_to_dingbats_1b_2460[] = {
/* U+2460 */ (char)0xAC,
/* U+2461 */ (char)0xAD,
/* U+2462 */ (char)0xAE,
/* U+2463 */ (char)0xAF,
/* U+2464 */ (char)0xB0,
/* U+2465 */ (char)0xB1,
/* U+2466 */ (char)0xB2,
/* U+2467 */ (char)0xB3,
/* U+2468 */ (char)0xB4,
/* U+2469 */ (char)0xB5,

static const char unicode_to_dingbats_1b_25A0[] = {
/* U+25A0 */ 0x6E,
/* U+25B2 */ 0x73,
/* U+25BC */ 0x74,
/* U+25CF */ 0x6C,
/* U+25D7 */ 0x77,
/* U+25C6 */ 0x77,
/* U+25D8 */ 0x78,
/* U+25D9 */ 0x79,
/* U+25DA */ 0x7A,
/* U+25DB */ 0x7B,
/* U+25DC */ 0x7C,
/* U+25DD */ 0x7D,
/* U+25DE */ 0x7E,
/* U+25DF */ 0x7F,
/* U+25E0 */ 0x80,
/* U+25E1 */ 0x81,
/* U+25E2 */ 0x82,
/* U+25E3 */ 0x83,
/* U+25E4 */ 0x84,
/* U+25E5 */ 0x85,
/* U+25E6 */ 0x86,
/* U+25E7 */ 0x87,
/* U+25E8 */ 0x88,
/* U+25E9 */ 0x89,
/* U+25EA */ 0x8A,
/* U+25EB */ 0x8B,
/* U+25EC */ 0x8C,
/* U+25ED */ 0x8D,
/* U+25EE */ 0x8E,
/* U+25EF */ 0x8F,
/* U+25F0 */ 0x90,
/* U+25F1 */ 0x91,
/* U+25F2 */ 0x92,
/* U+25F3 */ 0x93,
/* U+25F4 */ 0x94,
/* U+25F5 */ 0x95,
/* U+25F6 */ 0x96,
/* U+25F7 */ 0x97,
/* U+25F8 */ 0x98,
/* U+25F9 */ 0x99,
/* U+25FA */ 0x9A,
/* U+25FB */ 0x9B,
/* U+25FC */ 0x9C,
/* U+25FD */ 0x9D,
/* U+25FE */ 0x9E,
/* U+25FF */ 0x9F,
/* U+2600 */ 0xA0,
/* U+2601 */ 0xA1,
/* U+2602 */ 0xA2,
/* U+2603 */ 0xA3,
/* U+2604 */ 0xA4,
/* U+2605 */ 0xA5,
/* U+2606 */ 0xA6,
/* U+2607 */ 0xA7,
/* U+2608 */ 0xA8,
/* U+2609 */ 0xA9,
/* U+260A */ 0xAA,
/* U+260B */ 0xAB,
/* U+260C */ 0xAC,
/* U+260D */ 0xAD,
/* U+260E */ 0xAE,
/* U+260F */ 0xAF,
/* U+2610 */ 0xB0,
/* U+2611 */ 0xB1,
/* U+2612 */ 0xB2,
/* U+2613 */ 0xB3,
/* U+2614 */ 0xB4,
/* U+2615 */ 0xB5,
/* U+2616 */ 0xB6,
/* U+2617 */ 0xB7,
/* U+2618 */ 0xB8,
/* U+2619 */ 0xB9,
/* U+261A */ 0xBA,
/* U+261B */ 0xBB,
/* U+261C */ 0xBC,
/* U+261D */ 0xBD,
/* U+261E */ 0xBE,
/* U+261F */ 0xBF,
/* U+2620 */ 0xC0,
/* U+2621 */ 0xC1,
/* U+2622 */ 0xC2,
/* U+2623 */ 0xC3,
/* U+2624 */ 0xC4,
/* U+2625 */ 0xC5,
/* U+2626 */ 0xC6,
/* U+2627 */ 0xC7,
/* U+2628 */ 0xC8,
/* U+2629 */ 0xC9,
/* U+262A */ 0xCA,
/* U+262B */ 0xCB,
/* U+262C */ 0xCC,
/* U+262D */ 0xCD,
/* U+262E */ 0xCE,
/* U+262F */ 0xCF,
/* U+2630 */ 0xD0,
/* U+2631 */ 0xD1,
/* U+2632 */ 0xD2,
/* U+2633 */ 0xD3,
/* U+2634 */ 0xD4,
/* U+2635 */ 0xD5,
/* U+2636 */ 0xD6,
/* U+2637 */ 0xD7,
/* U+2638 */ 0xD8,
/* U+2639 */ 0xD9,
/* U+263A */ 0xDA,
/* U+263B */ 0xDB,
/* U+263C */ 0xDC,
/* U+263D */ 0xDD,
/* U+263E */ 0xDE,
/* U+263F */ 0xDF,
/* U+2640 */ 0xE0,
/* U+2641 */ 0xE1,
/* U+2642 */ 0xE2,
/* U+2643 */ 0xE3,
/* U+2644 */ 0xE4,
/* U+2645 */ 0xE5,
/* U+2646 */ 0xE6,
/* U+2647 */ 0xE7,
/* U+2648 */ 0xE8,
/* U+2649 */ 0xE9,
/* U+264A */ 0xEA,
/* U+264B */ 0xEB,
/* U+264C */ 0xEC,
/* U+264D */ 0xED,
/* U+264E */ 0xEE,
/* U+264F */ 0xEF,
/* U+2650 */ 0xF0,
/* U+2651 */ 0xF1,
/* U+2652 */ 0xF2,
/* U+2653 */ 0xF3,
/* U+2654 */ 0xF4,
/* U+2655 */ 0xF5,
/* U+2656 */ 0xF6,
/* U+2657 */ 0xF7,
/* U+2658 */ 0xF8,
/* U+2659 */ 0xF9,
/* U+265A */ 0xFA,
/* U+265B */ 0xFB,
/* U+265C */ 0xFC,
/* U+265D */ 0xFD,
/* U+265E */ 0xFE,
/* U+265F */ 0xFF,
};
00220 0x00,
00221 0x00,
00222 0x00,
00223 0x00,
00224 0x00,
00225 0x00,
00226 0x00,
00227 0x00,
00228 0x00,
00229 0x00,
00230 0x00,
00231 0x00,
00232 0x00,
00233 0x00,
00234 0x00,
00235 0x00,
00236 0x00,
00237 0x00,
00238 0x00,
00239 0x00,
00240 0x00,
00241 0x00,
00242 0x00,
00243 0x00,
00244 0x00,
00245 0x00,
00246 0x00,
00247 0x00,
00248 0x00,
00249 0x00,
00250 0x00,
00251 0x00,
00252 0x00,
00253 0x00,
00254 0x00,
00255 0x00,
00256 0x00,
00257 /* U+2605 */ 0x48,
00258 0x00,
00259 0x00,
00260 0x00,
00261 0x00,
00262 0x00,
00263 0x00,
00264 0x00,
00265 0x00,
00266 /* U+261E */ 0x2B,
00267 0x00,
00268 0x00,
00269 0x00,
00270 0x00,
00271 0x00,
00272 0x00,
00273 0x00,
00274 0x00,
00275 0x00,
00276 0x00,
00277 0x00,
00278 0x00,
00279 /* U+2618 */ 0x2A,
00280 0x00,
00281 0x00,
00282 /* U+2616 */ 0x2B,
00283 0x00,
00284 0x00,
00285 0x00,
00286 0x00,
00287 0x00,
00288 0x00,
00289 0x00,
00290 0x00,
00291 0x00,
00292 0x00,
00293 0x00,
00294 0x00,
00295 0x00,
00296 0x00,
00297 0x00,
00298 0x00,
00299 0x00,
00300 0x00,
00301 0x00,
00302 0x00,
00303 0x00,
00304 0x00,
00305 0x00,
00306 0x00,
00307 0x00,
00308 0x00,
00309 0x00,
00310 0x00,
00311 0x00,
00312 0x00,
00313 0x00,
00314 0x00,
00315 0x00,
00316 0x00,
00317 0x00,
00318 0x00,
00319 0x00,
00320 0x00,
00321 0x00,
00322 0x00,
00323 0x00,
00324 0x00,
00325 0x00,
00326 0x00,
00327 0x00,
00328 0x00,
00329 0x00,
00330 0x00,
00331 0x00,
00332 0x00,
00333 0x00,
00334 0x00,
00335 0x00,
00336 0x00,
00337 0x00,
00338 0x00,
00339 0x00,
00340 0x00,
00341 0x00,
00342 0x00,
00343 0x00,
00344 0x00,
00345 0x00,
00346 0x00,
00347 0x00,
00348 /* U+2660 */ (char)0xAB,
00349 0x00,
00350 0x00,
00351 /* U+2663 */ (char)0xA8,
00352 0x00,
00353 /* U+2665 */ (char)0xAA,
00354 /* U+2666 */ (char)0xA9,
00355 }
00356
00357 static const char unicode_to_dingbats_1b_2701[] = {
00358 /* U+2701 */ 0x21,
00359 /* U+2702 */ 0x22,
00360 /* U+2703 */ 0x23,
00361 /* U+2704 */ 0x24,
00362 0x00,
00363 /* U+2706 */ 0x26,
00364 /* U+2707 */ 0x27,
00365 /* U+2708 */ 0x28,
00366 /* U+2709 */ 0x29,
00367 0x00,
00368 0x00,
00369 /* U+270C */ 0x2C,
00370 /* U+270D */ 0x2D,
00371 /* U+270E */ 0x2E,
00372 /* U+270F */ 0x2F,
00373 /* U+2710 */ 0x30,
00374 /* U+2711 */ 0x31,
00375 /* U+2712 */ 0x32,
00376 /* U+2713 */ 0x33,
00377 /* U+2714 */ 0x34,
00378 /* U+2715 */ 0x35,
00379 /* U+2716 */ 0x36,
00380 /* U+2717 */ 0x37,
00381 /* U+2718 */ 0x38,
00382 /* U+2719 */ 0x39,
00383 /* U+271A */ 0x3A,
00384 /* U+271B */ 0x3B,
00385 /* U+271C */ 0x3C,
00386 /* U+271D */ 0x3D,
00387 /* U+271E */ 0x3E,
00388 /* U+271F */ 0x3F,
00389 /* U+2720 */ 0x40,
00390 /* U+2721 */ 0x41,
00391 /* U+2722 */ 0x42,
00392 /* U+2723 */ 0x43,
00393 /* U+2724 */ 0x44,
00394 /* U+2725 */ 0x45,
00395 /* U+2726 */ 0x46,
00396 /* U+2727 */ 0x47,
00397 0x00,
00398 /* U+2729 */ 0x49,
00399 /* U+272A */ 0x4A,
00400 /* U+272B */ 0x4B,
00401 /* U+272C */ 0x4C,
00402 /* U+272D */ 0x4D,
00403 /* U+272E */ 0x4E,
00404 /* U+272F */ 0x4F,
00405 /* U+2730 */ 0x50,
00406 /* U+2731 */ 0x51,
00407 /* U+2732 */ 0x52,
00408 /* U+2733 */ 0x53,
00409 /* U+2734 */ 0x54,
00410 /* U+2735 */ 0x55,
00411 /* U+2736 */ 0x56,
00412 /* U+2737 */ 0x57,
00413 /* U+2738 */ 0x58,
00414 /* U+2739 */ 0x59,
00415 /* U+273A */ 0x5A,
00416 /* U+273B */ 0x5B,
00417 /* U+273C */ 0x5C,
00418 /* U+273D */ 0x5D,
00419 /* U+273E */ 0x5E,
00420 /* U+273F */ 0x5F,
00421 /* U+2740 */ 0x60,
00422 /* U+2741 */ 0x61,
00423 /* U+2742 */ 0x62,
00424 /* U+2743 */ 0x63,
00425 /* U+2744 */ 0x64,
00426 /* U+2745 */ 0x65,
00427 /* U+2746 */ 0x66,
00428 /* U+2747 */ 0x67,
00429 /* U+2748 */ 0x68,
00430 /* U+2749 */ 0x69,
00431 /* U+274A */ 0x6A,
00432 /* U+274B */ 0x6B,
00433 0x00,
00434 /* U+274D */ 0x6D,
00435 0x00,
00436 /* U+274F */ 0x6F,
00437 /* U+2750 */ 0x70,
00438 /* U+2751 */ 0x71,
00439 /* U+2752 */ 0x72,
00440 0x00,
00441 0x00,
00442 0x00,
00443 /* U+2756 */ 0x76,
00444 0x00,
00445 /* U+2758 */ 0x78,
00446 /* U+2759 */ 0x79,
00447 /* U+275A */ 0x7A,
00448 /* U+275B */ 0x7B,
00449 /* U+275C */ 0x7C,
00450 /* U+275D */ 0x7D,
00451 /* U+275E */ 0x7E,
00452 0x00,
00453 0x00,
00454 /* U+2761 */ (char)0xA1,
00455 /* U+2762 */ (char)0xA2,
00456 /* U+2763 */ (char)0xA3,
00457 /* U+2764 */ (char)0xA4,
00458 /* U+2765 */ (char)0xA5,
00459 /* U+2766 */ (char)0xA6,
00460 /* U+2767 */ (char)0xA7,
00461 0x00,
00462 0x00,
00463 0x00,
00464 0x00,
00465 0x00,
00466 0x00,
00467 0x00,
00468 0x00,
00469 0x00,
00470 0x00,
00471 0x00,
00472 0x00,
00473 0x00,
00474 0x00,
00475 /* U+2776 */ (char)0xB6,
00476 /* U+2777 */ (char)0xB7,
00477 /* U+2778 */ (char)0xB8,
00478 /* U+2779 */ (char)0xB9,
00479 /* U+277A */ (char)0xBA,
00480 /* U+277B */ (char)0xBB,
static const char unicode_to_dingbats_1b_F8D7[] = {
    /* U+F8D7 */ (char)0x80,
    /* U+F8D8 */ (char)0x81,
    /* U+F8D9 */ (char)0x82,
    /* U+F8DA */ (char)0x83,
    /* U+F8DB */ (char)0x84,
    /* U+F8DC */ (char)0x85,
    /* U+F8DD */ (char)0x86,
    /* U+F8DE */ (char)0x87,
    /* U+F8DF */ (char)0x88,
    /* U+F8E0 */ (char)0x89,
    /* U+F8E1 */ (char)0x8A,
    /* U+F8E2 */ (char)0x8B,
    /* U+F8E3 */ (char)0x8C,
    /* U+F8E4 */ (char)0x8D,
    /* U+F8E5 */ (char)0x8E,
    /* U+F8E6 */ (char)0x8F,
    /* U+F8E7 */ (char)0x90,
    /* U+F8E8 */ (char)0x91,
    /* U+F8E9 */ (char)0x92,
    /* U+F8EA */ (char)0x93,
    /* U+F8EB */ (char)0x94,
    /* U+F8EC */ (char)0x95,
    /* U+F8ED */ (char)0x96,
    /* U+F8EE */ (char)0x97,
    /* U+F8EF */ (char)0x98,
    /* U+F8F0 */ (char)0x99,
    /* U+F8F1 */ (char)0x9A,
    /* U+F8F2 */ (char)0x9B,
    /* U+F8F3 */ (char)0x9C,
    /* U+F8F4 */ (char)0x9D,
    /* U+F8F5 */ (char)0x9E,
    /* U+F8F6 */ (char)0x9F,
    /* U+F8F7 */ (char)0xA0,
    /* U+F8F8 */ (char)0xA1,
    /* U+F8F9 */ (char)0xA2,
    /* U+F8FA */ (char)0xA3,
    /* U+F8FB */ (char)0xA4,
    /* U+F8FC */ (char)0xA5,
    /* U+F8FD */ (char)0xA6,
    /* U+F8FE */ (char)0xA7,
    /* U+F8FF */ (char)0xA8
};
static const unsigned short ucs_table_0300[] = {
    U+0300 0x0060,
    U+0301 0x00B4,
    U+0302 0x005E,
    U+0303 0x02DC,
    U+0304 0x00AF,
    U+0305 0x203E,
    U+0306 0x02D8,
    U+0307 0x02D9,
    U+0308 0x00A8,
    U+0309 0x0309,
    U+030A 0x02DA,
    U+030B 0x02DD,
    U+030C 0x030C,
    U+030D 0x030D,
    U+030E 0x030E,
    U+030F 0x030F,
    U+0310 0x0310,
    U+0311 0x0311,
    U+0312 0x0312,
    U+0313 0x1FBD,
    U+0314 0x1FFE,
    U+0315 0x0315,
    U+0316 0x0316,
    U+0317 0x0317,
    U+0318 0x0318,
    U+0319 0x0319,
    U+031A 0x031A,
    U+031B 0x031B,
    U+031C 0x031C,
    U+031D 0x031D,
    U+031E 0x031E,
    U+031F 0x031F,
    U+0320 0x0320,
    U+0321 0x0321,
    U+0322 0x0322,
    U+0323 0x0323,
    U+0324 0x0324,
    U+0325 0x0325,
    U+0326 0x0326,
    U+0327 0x0327,
    U+0328 0x0328,
    U+0329 0x0329,
    U+032A 0x032A,
    U+032B 0x032B,
    U+032C 0x032C,
    U+032D 0x032D,
    U+032E 0x032E,
    U+032F 0x032F,
    U+0330 0x0330,
    U+0331 0x0331,
    U+0332 0x0332,
    U+0333 0x0333,
    U+0334 0x0334,
    U+0335 0x0335,
    U+0336 0x0336,
    U+0337 0x0337,
    U+0338 0x0338,
    U+0339 0x0339,
    U+033A 0x033A,
    U+033B 0x033B,
    U+033C 0x033C,
    U+033D 0x033D,
    U+033E 0x033E,
    U+033F 0x033F,
    U+0340 0x0340,
    U+0341 0x0341,
    U+0342 0x0342,
    U+0343 0x0343,
    U+0344 0x0344,
    U+0345 0x0345,
    \(
};
static const unsigned short ucs_table_0483[] = {
  /* U+0483 */ 0x0483,
  /* U+0484 */ 0x0484,
  /* U+0485 */ 0x0485,
  /* U+0486 */ 0x0486,
  /* U+0487 */ 0x0487,
  /* U+0488 */ 0x0488,
  /* U+0489 */ 0x0489,
  /* U+048A */ 0x048A,
  /* U+048B */ 0x048B,
  /* U+048C */ 0x048C,
  /* U+048D */ 0x048D,
  /* U+048E */ 0x048E,
  /* U+048F */ 0x048F,
};

static const unsigned short ucs_table_0591[] = {
  /* U+0591 */ 0x0591,
  /* U+0592 */ 0x0592,
  /* U+0593 */ 0x0593,
  /* U+0594 */ 0x0594,
  /* U+0595 */ 0x0595,
  /* U+0596 */ 0x0596,
  /* U+0597 */ 0x0597,
  /* U+0598 */ 0x0598,
  /* U+0599 */ 0x0599,
  /* U+059A */ 0x059A,
  /* U+059B */ 0x059B,
  /* U+059C */ 0x059C,
  /* U+059D */ 0x059D,
  /* U+059E */ 0x059E,
  /* U+059F */ 0x059F,
  /* U+05A0 */ 0x05A0,
  /* U+05A1 */ 0x05A1,
  /* U+05A2 */ 0x05A2,
  /* U+05A3 */ 0x05A3,
  /* U+05A4 */ 0x05A4,
  /* U+05A5 */ 0x05A5,
  /* U+05A6 */ 0x05A6,
  /* U+05A7 */ 0x05A7,
  /* U+05A8 */ 0x05A8,
  /* U+05A9 */ 0x05A9,
  /* U+05AA */ 0x05AA,
  /* U+05AB */ 0x05AB,
  /* U+05AC */ 0x05AC,
  /* U+05AD */ 0x05AD,
  /* U+05AE */ 0x05AE,
  /* U+05AF */ 0x05AF,
  /* U+05B0 */ 0x05B0,
  /* U+05B1 */ 0x05B1,
  /* U+05B2 */ 0x05B2,
  /* U+05B3 */ 0x05B3,
  /* U+05B4 */ 0x05B4,
  /* U+05B5 */ 0x05B5,
  /* U+05B6 */ 0x05B6,
  /* U+05B7 */ 0x05B7,
  /* U+05B8 */ 0x05B8,
  /* U+05B9 */ 0x05B9,
  /* U+05BA */ 0x05BA,
  /* U+05BB */ 0x05BB,
  /* U+05BC */ 0x05BC,
  /* U+05BD */ 0x05BD,
  /* U+05BE */ 0x05BE,
  /* U+05BF */ 0x05BF,
};

static const unsigned short ucs_table_064B[] = {
  /* U+064B */ 0xFE70,
  /* U+064C */ 0xFE72,
  /* U+064D */ 0xFE74,
  /* U+064E */ 0xFE76,
  /* U+064F */ 0xFE78,
  /* U+0650 */ 0xFE7A,
};
00173 /* U+0651 */ 0xFE7C,
00174 /* U+0652 */ 0xFE7E,
00175 0x00,
00176 0x00,
00177 0x00,
00178 0x00,
00179 0x00,
00180 0x00,
00181 0x00,
00182 0x00,
00183 0x00,
00184 0x00,
00185 0x00,
00186 0x00,
00187 0x00,
00188 0x00,
00189 0x00,
00190 0x00,
00191 0x00,
00192 0x00,
00193 0x00,
00194 0x00,
00195 0x00,
00196 0x00,
00197 0x00,
00198 0x00,
00199 0x00,
00200 0x00,
00201 0x00,
00202 0x00,
00203 0x00,
00204 /* U+0670 */ 0x0670,
00205 0x00,
00206 0x00,
00207 0x00,
00208 0x00,
00209 0x00,
00210 0x00,
00211 0x00,
00212 0x00,
00213 0x00,
00214 0x00,
00215 0x00,
00216 0x00,
00217 0x00,
00218 0x00,
00219 0x00,
00220 0x00,
00221 0x00,
00222 0x00,
00223 0x00,
00224 0x00,
00225 0x00,
00226 0x00,
00227 0x00,
00228 0x00,
00229 0x00,
00230 0x00,
00231 0x00,
00232 0x00,
00233 0x00,
00234 0x00,
00235 0x00,
00236 0x00,
00237 0x00,
00238 0x00,
00239 0x00,
00240 0x00,
00241 0x00,
00242 0x00,
00243 0x00,
00244 0x00,
00245 0x00,
00246 0x00,
00247 0x00,
00248 0x00,
00249 0x00,
00250 0x00,
00251 0x00,
00252 0x00,
00253 0x00,
00254 0x00,
00255 0x00,
00256 0x00,
00257 0x00,
00258 0x00,
00259 0x00,
00260 0x00,
00261 0x00,
00262 0x00,
00263 0x00,
00264 0x00,
00265 0x00,
00266 0x00,
00267 0x00,
00268 0x00,
00269 0x00,
00270 0x00,
00271 0x00,
00272 0x00,
00273 0x00,
00274 0x00,
00275 0x00,
00276 0x00,
00277 0x00,
00278 0x00,
00279 0x00,
00280 0x00,
00281 0x00,
00282 0x00,
00283 0x00,
00284 0x00,
00285 0x00,
00286 0x00,
00287 0x00,
00288 0x00,
00289 0x00,
00290 0x00,
00291 0x00,
00292 0x00,
00293 0x00,
00294 0x00,
00295 0x00,
00296 0x00,
00297 0x00,
00298 0x00,
00299 0x00,
00300 0x00,
00301 0x00,
00302 0x00,
00303 0x00,
00304 0x00,
00305 0x00,
00306 /* U+06D6 */ 0x06D6,
00307 /* U+06D7 */ 0x06D7,
00308 /* U+06D8 */ 0x06D8,
00309 /* U+06D9 */ 0x06D9,
00310 /* U+06DA */ 0x06DA,
00311 /* U+06DB */ 0x06DB,
00312 /* U+06DC */ 0x06DC,
00313 0x00,
00314 0x00,
00315 /* U+06DF */ 0x06DF,
00316 /* U+06E0 */ 0x06E0,
00317 /* U+06E1 */ 0x06E1,
00318 /* U+06E2 */ 0x06E2,
00319 /* U+06E3 */ 0x06E3,
00320 /* U+06E4 */ 0x06E4,
00321 0x00,
00322 0x00,
00323 /* U+06E7 */ 0x06E7,
00324 /* U+06E8 */ 0x06E8,
00325 0x00,
00326 /* U+06EA */ 0x06EA,
00327 /* U+06EB */ 0x06EB,
00328 /* U+06EC */ 0x06EC,
00329 /* U+06ED */ 0x06ED,
00330 });
00331 static const unsigned short ucs_table_0901[] = {
00332   /* U+0901 */ 0x0901,
00333   /* U+0902 */ 0x0902,
00334   0x00,
00335   0x00,
00336   0x00,
00337   0x00,
00338   0x00,
00347 0x00,
00348 0x00,
00349 0x00,
00350 0x00,
00351 0x00,
00352 0x00,
00353 0x00,
00354 0x00,
00355 0x00,
00356 0x00,
00357 0x00,
00358 0x00,
00359 0x00,
00360 0x00,
00361 0x00,
00362 0x00,
00363 0x00,
00364 0x00,
00365 0x00,
00366 0x00,
00367 0x00,
00368 0x00,
00369 0x00,
00370 0x00,
00371 0x00,
00372 0x00,
00373 0x00,
00374 0x00,
00375 0x00,
00376 0x00,
00377 0x00,
00378 0x00,
00379 0x00,
00380 0x00,
00381 0x00,
00382 0x00,
00383 0x00,
00384 0x00,
00385 0x00,
00386 0x00,
00387 0x00,
00388 0x00,
00389 0x00,
00390 0x00,
00391 0x00,
00392 /* U+093C */ 0x093C,
00393 0x00,
00394 0x00,
00395 0x00,
00396 0x00,
00397 /* U+0941 */ 0x0941,
00398 /* U+0942 */ 0x0942,
00399 /* U+0943 */ 0x0943,
00400 /* U+0944 */ 0x0944,
00401 /* U+0945 */ 0x0945,
00402 /* U+0946 */ 0x0946,
00403 /* U+0947 */ 0x0947,
00404 /* U+0948 */ 0x0948,
00405 0x00,
00406 0x00,
00407 0x00,
00408 0x00,
00409 /* U+094D */ 0x094D,
00410 0x00,
00411 0x00,
00412 0x00,
00413 /* U+0951 */ 0x0951,
00414 /* U+0952 */ 0x0952,
00415 /* U+0953 */ 0x0953,
00416 /* U+0954 */ 0x0954,
00417 0x00,
00418 0x00,
00419 0x00,
00420 0x00,
00421 0x00,
00422 0x00,
00423 0x00,
00424 0x00,
00425 0x00,
00426 0x00,
00427 0x00,
00428 0x00,
00429 0x00,
00430 /* U+0962 */ 0x0962,
00431 /* U+0963 */ 0x0963,
00432 0x00,
00433 0x00,
00434 0x00,
00435 0x00,
00436 0x00,
00437 0x00,
00438 0x00,
00439 0x00,
00440 0x00,
00441 0x00,
00442 0x00,
00443 0x00,
00444 0x00,
00445 0x00,
00446 0x00,
00447 0x00,
00448 0x00,
00449 0x00,
00450 0x00,
00451 0x00,
00452 0x00,
00453 0x00,
00454 0x00,
00455 0x00,
00456 0x00,
00457 0x00,
00458 0x00,
00459 0x00,
00460 0x00,
00461 /* U+0981 */ 0x0981,
00462 0x00,
00463 0x00,
00464 0x00,
00465 0x00,
00466 0x00,
00467 0x00,
00468 0x00,
00469 0x00,
00470 0x00,
00471 0x00,
00472 0x00,
00473 0x00,
00474 0x00,
00475 0x00,
00476 0x00,
00477 0x00,
00478 0x00,
00479 0x00,
00480 0x00,
00481 0x00,
00482 0x00,
00483 0x00,
00484 0x00,
00485 0x00,
00486 0x00,
00487 0x00,
00488 0x00,
00489 0x00,
00490 0x00,
00491 0x00,
00492 0x00,
00493 0x00,
00494 0x00,
00495 0x00,
00496 0x00,
00497 0x00,
00498 0x00,
00499 0x00,
00500 0x00,
00501 0x00,
00502 0x00,
00503 0x00,
00504 0x00,
00505 0x00,
00506 0x00,
00507 0x00,
00508 0x00,
00509 0x00,
00510 0x00,
00511 0x00,
00512 0x00,
00513 0x00,
00514 0x00,
00515 0x00,
00516 0x00,
00517 0x00,
00518 0x00,
00519 0x00,
00520 /* U+09BC */ 0x09BC,
00521 0x00,
00522 0x00,
00523 0x00,
00524 0x00,
00525 /* U+09C1 */ 0x09C1,
00526 /* U+09C2 */ 0x09C2,
00527 /* U+09C3 */ 0x09C3,
00528 /* U+09C4 */ 0x09C4,
00529 0x00,
00530 0x00,
00531 0x00,
00532 0x00,
00533 0x00,
00534 0x00,
00535 0x00,
00536 0x00,
00537 /* U+09CD */ 0x09CD,
00538 0x00,
00539 0x00,
00540 0x00,
00541 0x00,
00542 0x00,
00543 0x00,
00544 0x00,
00545 0x00,
00546 0x00,
00547 0x00,
00548 0x00,
00549 0x00,
00550 0x00,
00551 0x00,
00552 0x00,
00553 0x00,
00554 0x00,
00555 0x00,
00556 0x00,
00557 0x00,
00558 /* U+09E2 */ 0x09E2,
00559 /* U+09E3 */ 0x09E3,
00560 0x00,
00561 0x00,
00562 0x00,
00563 0x00,
00564 0x00,
00565 0x00,
00566 0x00,
00567 0x00,
00568 0x00,
00569 0x00,
00570 0x00,
00571 0x00,
00572 0x00,
00573 0x00,
00574 0x00,
00575 0x00,
00576 0x00,
00577 0x00,
00578 0x00,
00579 0x00,
00580 0x00,
00581 0x00,
00582 0x00,
00583 0x00,
00584 0x00,
00585 0x00,
00586 0x00,
00587 0x00,
00588 0x00,
00589 0x00,
00590 /* U+0A02 */ 0x0A02,
00591 0x00,
00592 0x00,
00593 0x00,
00594 0x00,
00595 0x00,
00596 0x00,
00597 0x00,
00598 0x00,
00599 0x00,
00600 0x00,
00601 0x00,
00602 0x00,
00603 0x00,
00604 0x00,
00605 0x00,
00606 0x00,
00607 0x00,
00695 0x00,
00696 0x00,
00697 0x00,
00698 0x00,
00699 0x00,
00700 /* U+0A70 */ 0xA70,
00701 /* U+0A71 */ 0xA71,
00702 0x00,
00703 0x00,
00704 0x00,
00705 0x00,
00706 0x00,
00707 0x00,
00708 0x00,
00709 0x00,
00710 0x00,
00711 0x00,
00712 0x00,
00713 0x00,
00714 0x00,
00715 0x00,
00716 0x00,
00717 /* U+0A81 */ 0xA81,
00718 /* U+0A82 */ 0xA82,
00719 0x00,
00720 0x00,
00721 0x00,
00722 0x00,
00723 0x00,
00724 0x00,
00725 0x00,
00726 0x00,
00727 0x00,
00728 0x00,
00729 0x00,
00730 0x00,
00731 0x00,
00732 0x00,
00733 0x00,
00734 0x00,
00735 0x00,
00736 0x00,
00737 0x00,
00738 0x00,
00739 0x00,
00740 0x00,
00741 0x00,
00742 0x00,
00743 0x00,
00744 0x00,
00745 0x00,
00746 0x00,
00747 0x00,
00748 0x00,
00749 0x00,
00750 0x00,
00751 0x00,
00752 0x00,
00753 0x00,
00754 0x00,
00755 0x00,
00756 0x00,
00757 0x00,
00758 0x00,
00759 0x00,
00760 0x00,
00761 0x00,
00762 0x00,
00763 0x00,
00764 0x00,
00765 0x00,
00766 0x00,
00767 0x00,
00768 0x00,
00769 0x00,
00770 0x00,
00771 0x00,
00772 0x00,
00773 0x00,
00774 0x00,
00775 0x00,
00776 /* U+0ABC */ 0xABCD,
00777 0x00,
00778 0x00,
00779 0x00,
00780 0x00,
00781 /* U+0AC1 */ 0xAC1,
00782 /* U+0AC2 */ 0x0AC2,
00783 /* U+0AC3 */ 0x0AC3,
00784 /* U+0AC4 */ 0x0AC4,
00785 /* U+0AC5 */ 0x0AC5,
00786 0x00,
00787 /* U+0AC7 */ 0x0AC7,
00788 /* U+0AC8 */ 0x0AC8,
00789 0x00,
00790 0x00,
00791 0x00,
00792 0x00,
00793 /* U+0ACD */ 0x0ACD,
00794 0x00,
00795 0x00,
00796 0x00,
00797 0x00,
00798 0x00,
00799 0x00,
00800 0x00,
00801 0x00,
00802 0x00,
00803 0x00,
00804 0x00,
00805 0x00,
00806 0x00,
00807 0x00,
00808 0x00,
00809 0x00,
00810 0x00,
00811 0x00,
00812 0x00,
00813 0x00,
00814 0x00,
00815 0x00,
00816 0x00,
00817 0x00,
00818 0x00,
00819 0x00,
00820 0x00,
00821 0x00,
00822 0x00,
00823 0x00,
00824 0x00,
00825 0x00,
00826 0x00,
00827 0x00,
00828 0x00,
00829 0x00,
00830 0x00,
00831 0x00,
00832 0x00,
00833 0x00,
00834 0x00,
00835 0x00,
00836 0x00,
00837 0x00,
00838 0x00,
00839 0x00,
00840 0x00,
00841 0x00,
00842 0x00,
00843 0x00,
00844 0x00,
00845 /* U+0B01 */ 0x0B01,
00846 0x00,
00847 0x00,
00848 0x00,
00849 0x00,
00850 0x00,
00851 0x00,
00852 0x00,
00853 0x00,
00854 0x00,
00855 0x00,
00856 0x00,
00857 0x00,
00858 0x00,
00859 0x00,
00860 0x00,
00861 0x00,
00862 0x00,
00863 0x00,
00864 0x00,
00865 0x00,
00866 0x00,
00867 0x00,
00868 0x00,
Generated by Doxygen
Generated by Doxygen
01043 0x00,
01044 0x00,
01045 0x00,
01046 0x00,
01047 0x00,
01048 0x00,
01049 /* U+0BCD */ 0x0BCD,
01050 0x00,
01051 0x00,
01052 0x00,
01053 0x00,
01054 0x00,
01055 0x00,
01056 0x00,
01057 0x00,
01058 0x00,
01059 0x00,
01060 0x00,
01061 0x00,
01062 0x00,
01063 0x00,
01064 0x00,
01065 0x00,
01066 0x00,
01067 0x00,
01068 0x00,
01069 0x00,
01070 0x00,
01071 0x00,
01072 0x00,
01073 0x00,
01074 0x00,
01075 0x00,
01076 0x00,
01077 0x00,
01078 0x00,
01079 0x00,
01080 0x00,
01081 0x00,
01082 0x00,
01083 0x00,
01084 0x00,
01085 0x00,
01086 0x00,
01087 0x00,
01088 0x00,
01089 0x00,
01090 0x00,
01091 0x00,
01092 0x00,
01093 0x00,
01094 0x00,
01095 0x00,
01096 0x00,
01097 0x00,
01098 0x00,
01099 0x00,
01100 0x00,
01101 0x00,
01102 0x00,
01103 0x00,
01104 0x00,
01105 0x00,
01106 0x00,
01107 0x00,
01108 0x00,
01109 0x00,
01110 0x00,
01111 0x00,
01112 0x00,
01113 0x00,
01114 0x00,
01115 0x00,
01116 0x00,
01117 0x00,
01118 0x00,
01119 0x00,
01120 0x00,
01121 0x00,
01122 0x00,
01123 0x00,
01124 0x00,
01125 0x00,
01126 0x00,
01127 0x00,
01128 0x00,
01129 0x00,
01130 0x00,
01131 0x00,
01132 0x00,
01133 0x00,
01134 0x00,
01135 0x00,
01136 0x00,
01137 0x00,
01138 0x00,
01139 0x00,
01140 0x00,
01141 0x00,
01142 0x00,
01143 0x00,
01144 0x00,
01145 0x00,
01146 0x00,
01147 0x00,
01148 0x00,
01149 0x00,
01150 0x00,
01151 0x00,
01152 0x00,
01153 0x00,
01154 0x00,
01155 0x00,
01156 0x00,
01157 0x00,
01158 0x00,
01159 0x00,
01160 0x00,
01161 0x00,
01162 /* U+0C3E */ 0x0C3E,
01163 /* U+0C3F */ 0x0C3F,
01164 /* U+0C40 */ 0x0C40,
01165 0x00,
01166 0x00,
01167 0x00,
01168 0x00,
01169 0x00,
01170 /* U+0C46 */ 0x0C46,
01171 /* U+0C47 */ 0x0C47,
01172 /* U+0C48 */ 0x0C48,
01173 0x00,
01174 /* U+0C4A */ 0x0C4A,
01175 /* U+0C4B */ 0x0C4B,
01176 /* U+0C4C */ 0x0C4C,
01177 /* U+0C4D */ 0x0C4D,
01178 0x00,
01179 0x00,
01180 0x00,
01181 0x00,
01182 0x00,
01183 0x00,
01184 0x00,
01185 /* U+0C55 */ 0x0C55,
01186 /* U+0C56 */ 0x0C56,
01187 0x00,
01188 0x00,
01189 0x00,
01190 0x00,
01191 0x00,
01192 0x00,
01193 0x00,
01194 0x00,
01195 0x00,
01196 0x00,
01197 0x00,
01198 0x00,
01199 0x00,
01200 0x00,
01201 0x00,
01202 0x00,
01203 0x00,
01204 0x00,
01205 0x00,
01206 0x00,
01207 0x00,
01208 0x00,
01209 0x00,
01210 0x00,
01211 0x00,
01212 0x00,
01213 0x00,
01214 0x00,
01215 0x00,
01216 0x00,
01217 0x00,
01218 0x00,
01219 0x00,
01220 0x00,
01221 0x00,
01222 0x00,
01223 0x00,
01224 0x00,
01225 0x00,
01226 0x00,
01227 0x00,
01228 0x00,
01229 0x00,
01230 0x00,
01231 0x00,
01232 0x00,
01233 0x00,
01234 0x00,
01235 0x00,
01236 0x00,
01237 0x00,
01238 0x00,
01239 0x00,
01240 0x00,
01241 0x00,
01242 0x00,
01243 0x00,
01244 0x00,
01245 0x00,
01246 0x00,
01247 0x00,
01248 0x00,
01249 0x00,
01250 0x00,
01251 0x00,
01252 0x00,
01253 0x00,
01254 0x00,
01255 0x00,
01256 0x00,
01257 0x00,
01258 0x00,
01259 0x00,
01260 0x00,
01261 0x00,
01262 0x00,
01263 0x00,
01264 0x00,
01265 0x00,
01266 0x00,
01267 0x00,
01268 0x00,
01269 0x00,
01270 0x00,
01271 0x00,
01272 0x00,
01273 0x00,
01274 0x00,
01275 0x00,
01276 0x00,
01277 0x00,
01278 0x00,
01279 0x00,
01280 0x00,
01281 0x00,
01282 0x00,
01283 0x00,
01284 0x00,
01285 0x00,
01286 0x00,
01287 0x00,
01288 0x00,
01289 0x00,
01290 0x00,
01291 /* 0x0CBF */ 0x0CBF,
01304 /* U+0CCC */ 0x0CCC,
01305 /* U+0CCD */ 0x0CCD,
01306 0x00,
01307 0x00,
01308 0x00,
01309 0x00,
01310 0x00,
01311 0x00,
01312 0x00,
01313 0x00,
01314 0x00,
01315 0x00,
01316 0x00,
01317 0x00,
01318 0x00,
01319 0x00,
01320 0x00,
01321 0x00,
01322 0x00,
01323 0x00,
01324 0x00,
01325 0x00,
01326 0x00,
01327 0x00,
01328 0x00,
01329 0x00,
01330 0x00,
01331 0x00,
01332 0x00,
01333 0x00,
01334 0x00,
01335 0x00,
01336 0x00,
01337 0x00,
01338 0x00,
01339 0x00,
01340 0x00,
01341 0x00,
01342 0x00,
01343 0x00,
01344 0x00,
01345 0x00,
01346 0x00,
01347 0x00,
01348 0x00,
01349 0x00,
01350 0x00,
01351 0x00,
01352 0x00,
01353 0x00,
01354 0x00,
01355 0x00,
01356 0x00,
01357 0x00,
01358 0x00,
01359 0x00,
01360 0x00,
01361 0x00,
01362 0x00,
01363 0x00,
01364 0x00,
01365 0x00,
01366 0x00,
01367 0x00,
01368 0x00,
01369 0x00,
01370 0x00,
01371 0x00,
01372 0x00,
01373 0x00,
01374 0x00,
01375 0x00,
01376 0x00,
01377 0x00,
01378 0x00,
01379 0x00,
01380 0x00,
01381 0x00,
01382 0x00,
01383 0x00,
01384 0x00,
01385 0x00,
01386 0x00,
01387 0x00,
01388 0x00,
01389 0x00,
01390 0x00,
```c
01391 0x00,
01392 0x00,
01393 0x00,
01394 0x00,
01395 0x00,
01396 0x00,
01397 0x00,
01398 0x00,
01399 0x00,
01400 0x00,
01401 0x00,
01402 0x00,
01403 0x00,
01404 0x00,
01405 0x00,
01406 0x00,
01407 0x00,
01408 0x00,
01409 0x00,
01410 0x00,
01411 0x00,
01412 0x00,
01413 0x00,
01414 0x00,
01415 0x00,
01416 0x00,
01417 0x00,
01418 0x00,
01419 0x00,
01420 0x00,
01421 /* U+0D41 */ 0x0D41,
01422 /* U+0D42 */ 0x0D42,
01423 /* U+0D43 */ 0x0D43,
01424 0x00,
01425 0x00,
01426 0x00,
01427 0x00,
01428 0x00,
01429 0x00,
01430 0x00,
01431 0x00,
01432 0x00,
01433 /* U+0D4D */ 0x0D4D,
01434 );
01435 static const unsigned short ucs_table_0E31[] = {
01436 /* U+0E31 */ 0x0E31,
01437 /* U+0E33 */ 0x0E33,
01438 /* U+0E34 */ 0x0E34,
01439 /* U+0E35 */ 0x0E35,
01440 /* U+0E36 */ 0x0E36,
01441 /* U+0E37 */ 0x0E37,
01442 /* U+0E38 */ 0x0E38,
01443 /* U+0E39 */ 0x0E39,
01444 /* U+0E3A */ 0x0E3A,
01445 /* U+0E3B */ 0x0E3B,
01446 /* U+0E3C */ 0x0E3C,
01447 /* U+0E3D */ 0x0E3D,
01448 /* U+0E3E */ 0x0E3E,
01449 /* U+0E3F */ 0x0E3F,
01450 /* U+0E40 */ 0x0E40,
01451 /* U+0E41 */ 0x0E41,
01452 /* U+0E42 */ 0x0E42,
01453 /* U+0E43 */ 0x0E43,
01454 /* U+0E44 */ 0x0E44,
01455 /* U+0E45 */ 0x0E45,
01456 /* U+0E46 */ 0x0E46,
01457 /* U+0E47 */ 0x0E47,
01458 /* U+0E48 */ 0x0E48,
01459 /* U+0E49 */ 0x0E49,
01460 /* U+0E4A */ 0x0E4A,
01461 /* U+0E4B */ 0x0E4B,
01462 /* U+0E4C */ 0x0E4C,
01463 /* U+0E4D */ 0x0E4D,
01464 /* U+0E4E */ 0x0E4E,
01465 /* U+0E4F */ 0x0E4F,
01466 /* U+0E50 */ 0x0E50,
01467 /* U+0E51 */ 0x0E51,
01468 /* U+0E52 */ 0x0E52,
01469 /* U+0E53 */ 0x0E53,
01470 /* U+0E54 */ 0x0E54,
01471 /* U+0E55 */ 0x0E55,
01472 /* U+0E56 */ 0x0E56,
01473 /* U+0E57 */ 0x0E57,
01474 /* U+0E58 */ 0x0E58,
01475 /* U+0E59 */ 0x0E59,
01476 /* U+0E5A */ 0x0E5A,
01477 /* U+0E5B */ 0x0E5B,
```
01478 0x00,
01479 0x00,
01480 0x00,
01481 0x00,
01482 0x00,
01483 0x00,
01484 0x00,
01485 0x00,
01486 0x00,
01487 0x00,
01488 0x00,
01489 0x00,
01490 0x00,
01491 0x00,
01492 0x00,
01493 0x00,
01494 0x00,
01495 0x00,
01496 0x00,
01497 0x00,
01498 0x00,
01499 0x00,
01500 0x00,
01501 0x00,
01502 0x00,
01503 0x00,
01504 0x00,
01505 0x00,
01506 0x00,
01507 0x00,
01508 0x00,
01509 0x00,
01510 0x00,
01511 0x00,
01512 0x00,
01513 0x00,
01514 0x00,
01515 0x00,
01516 0x00,
01517 0x00,
01518 0x00,
01519 0x00,
01520 0x00,
01521 0x00,
01522 0x00,
01523 0x00,
01524 0x00,
01525 0x00,
01526 0x00,
01527 0x00,
01528 0x00,
01529 0x00,
01530 0x00,
01531 0x00,
01532 0x00,
01533 0x00,
01534 0x00,
01535 0x00,
01536 0x00,
01537 0x00,
01538 0x00,
01539 0x00,
01540 0x00,
01541 0x00,
01542 0x00,
01543 0x00,
01544 0x00,
01545 0x00,
01546 0x00,
01547 0x00,
01548 0x00,
01549 0x00,
01550 0x00,
01551 0x00,
01552 0x00,
01553 0x00,
01554 0x00,
01555 0x00,
01556 0x00,
01557 0x00,
01558 0x00,
01559 0x00,
01560 0x00,
01561 0x00,
01562 0x00,
01563 0x00,
01564 0x00,
01565 /* U+0EB1 */ 0xEB1,
01566 0x00,
01567 0x00,
01568 /* U+0EB4 */ 0xEB4,
01569 /* U+0EB5 */ 0xEB5,
01570 /* U+0EB6 */ 0xEB6,
01571 /* U+0EB7 */ 0xEB7,
01572 /* U+0EB8 */ 0xEB8,
01573 /* U+0EB9 */ 0xEB9,
01574 0x00,
01575 /* U+0EBB */ 0xEBB,
01576 /* U+0EBC */ 0xEBC,
01577 0x00,
01578 0x00,
01579 0x00,
01580 0x00,
01581 0x00,
01582 0x00,
01583 0x00,
01584 0x00,
01585 0x00,
01586 0x00,
01587 0x00,
01588 /* U+0EC8 */ 0xEC8,
01589 /* U+0EC9 */ 0xEC9,
01590 /* U+0ECA */ 0xECA,
01591 /* U+0ECB */ 0xECB,
01592 /* U+0ECC */ 0xECB,
01593 /* U+0EC6 */ 0xEC6,
01594 0x00,
01595 0x00,
01596 0x00,
01597 0x00,
01598 0x00,
01599 0x00,
01600 0x00,
01601 0x00,
01602 0x00,
01603 0x00,
01604 0x00,
01605 0x00,
01606 0x00,
01607 0x00,
01608 0x00,
01609 0x00,
01610 0x00,
01611 0x00,
01612 0x00,
01613 0x00,
01614 0x00,
01615 0x00,
01616 0x00,
01617 0x00,
01618 0x00,
01619 0x00,
01620 0x00,
01621 0x00,
01622 0x00,
01623 0x00,
01624 0x00,
01625 0x00,
01626 0x00,
01627 0x00,
01628 0x00,
01629 0x00,
01630 0x00,
01631 0x00,
01632 0x00,
01633 0x00,
01634 0x00,
01635 0x00,
01636 0x00,
01637 0x00,
01638 0x00,
01639 0x00,
01640 0x00,
01641 0x00,
01642 0x00,
01643 0x00,
01644 0x00,
01645 0x00,
01646 0x00,
01647 0x00,
01648 0x00,
01649 0x00,
01650 0x00,
01651 0x00,
01652 0x00,
01653 0x00,
01654 0x00,
01655 0x00,
01656 0x00,
01657 0x00,
01658 0x00,
01659 0x00,
01660 0x00,
01661 0x00,
01662 0x00,
01663 0x00,
01664 0x00,
01665 0x00,
01666 0x00,
01667 0x00,
01668 /* U+0F18 */ 0x0F18,
01669 /* U+0F19 */ 0x0F19,
01670 0x00,
01671 0x00,
01672 0x00,
01673 0x00,
01674 0x00,
01675 0x00,
01676 0x00,
01677 0x00,
01678 0x00,
01679 0x00,
01680 0x00,
01681 0x00,
01682 0x00,
01683 0x00,
01684 0x00,
01685 0x00,
01686 0x00,
01687 0x00,
01688 0x00,
01689 0x00,
01690 0x00,
01691 0x00,
01692 0x00,
01693 0x00,
01694 0x00,
01695 0x00,
01696 0x00,
01697 /* U+0F35 */ 0x0F35,
01698 0x00,
01699 /* U+0F37 */ 0x0F37,
01700 0x00,
01701 /* U+0F39 */ 0x0F39,
01702 0x00,
01703 0x00,
01704 0x00,
01705 0x00,
01706 0x00,
01707 0x00,
01708 0x00,
01709 0x00,
01710 0x00,
01711 0x00,
01712 0x00,
01713 0x00,
01714 0x00,
01715 0x00,
01716 0x00,
01717 0x00,
01718 0x00,
01719 0x00,
01720 0x00,
01721 0x00,
01722 0x00,
01723 0x00,
01724 0x00,
01725 0x00,
01726 0x00,
01727 0x00,
01728 0x00,
01729 0x00,
01730 0x00,
01731 0x00,
01732 0x00,
01733 0x00,
01734 0x00,
01735 0x00,
01736 0x00,
01737 0x00,
01738 0x00,
```c
00739 0x00,
01740 0x00,
01741 0x00,
01742 0x00,
01743 0x00,
01744 0x00,
01745 0x00,
01746 0x00,
01747 0x00,
01748 0x00,
01749 0x00,
01750 0x00,
01751 0x00,
01752 0x00,
01753 0x00,
01754 0x00,
01755 0x00,
01756 0x00,
01757 / U+0F71 / 0xF71,
01758 / U+0F72 / 0xF72,
01759 / U+0F73 / 0xF73,
01760 / U+0F74 / 0xF74,
01761 / U+0F75 / 0xF75,
01762 / U+0F76 / 0xF76,
01763 / U+0F77 / 0xF77,
01764 / U+0F78 / 0xF78,
01765 / U+0F79 / 0xF79,
01766 / U+0F7A / 0xF7A,
01767 / U+0F7B / 0xF7B,
01768 / U+0F7C / 0xF7C,
01769 / U+0F7D / 0xF7D,
01770 / U+0F7E / 0xF7E,
01771 0x00,
01772 / U+0F80 / 0xF80,
01773 / U+0F81 / 0xF81,
01774 / U+0F82 / 0xF82,
01775 / U+0F83 / 0xF83,
01776 / U+0F84 / 0xF84,
01777 0x00,
01778 / U+0F86 / 0xF86,
01779 / U+0F87 / 0xF87,
01780 0x00,
01781 0x00,
01782 0x00,
01783 0x00,
01784 0x00,
01785 0x00,
01786 0x00,
01787 0x00,
01788 / U+0F90 / 0xF90,
01789 / U+0F91 / 0xF91,
01790 / U+0F92 / 0xF92,
01791 / U+0F93 / 0xF93,
01792 / U+0F94 / 0xF94,
01793 / U+0F95 / 0xF95,
01794 0x00,
01795 / U+0F97 / 0xF97,
01796 0x00,
01797 / U+0F99 / 0xF99,
01798 / U+0F9A / 0xF9A,
01799 / U+0F9B / 0xF9B,
01800 / U+0F9C / 0xF9C,
01801 / U+0F9D / 0xF9D,
01802 / U+0F9E / 0xF9E,
01803 / U+0F9F / 0xF9F,
01804 / U+0FA0 / 0xFA0,
01805 / U+0FA1 / 0xFA1,
01806 / U+0FA2 / 0xFA2,
01807 / U+0FA3 / 0xFA3,
01808 / U+0FA4 / 0xFA4,
01809 / U+0FA5 / 0xFA5,
01810 / U+0FA6 / 0xFA6,
01811 / U+0FA7 / 0xFA7,
01812 / U+0FA8 / 0xFA8,
01813 / U+0FA9 / 0xFA9,
01814 / U+0FAA / 0xFAA,
01815 / U+0FAB / 0xFB0,
01816 / U+0FAC / 0xFB0,
01817 / U+0FAD / 0xFB0,
01818 0x00,
01819 0x00,
01820 0x00,
01821 / U+0FB1 / 0xFB1,
01822 / U+0FB2 / 0xFB2,
01823 / U+0FB3 / 0xFB3,
01824 / U+0FB4 / 0xFB4,
01825 / U+0FB5 / 0xFB5,
```
01826 /* U+0FB6 */ 0xFB6, 01827 /* U+0FB7 */ 0xFB7, 01828 0x00, 01829 /* U+0FB9 */ 0xFB9, 01830 */ 0x0FB9, 01831 0x00, 01832 static const unsigned short ucs_table_20D0[] = { 01833 /* U+20D0 */ 0x20D0, 01834 /* U+20D1 */ 0x20D1, 01835 /* U+20D2 */ 0x20D2, 01836 /* U+20D3 */ 0x20D3, 01837 /* U+20D4 */ 0x20D4, 01838 /* U+20D5 */ 0x20D5, 01839 /* U+20D6 */ 0x20D6, 01840 /* U+20D7 */ 0x20D7, 01841 /* U+20D8 */ 0x20D8, 01842 /* U+20D9 */ 0x20D9, 01843 /* U+20DA */ 0x20DA, 01844 /* U+20DB */ 0x20DB, 01845 /* U+20DC */ 0x20DC, 01846 0x00, 01847 0x00, 01848 0x00, 01849 0x00, 01850 /* U+20E1 */ 0x20E1, 01851 */ 0x20E1, 01852 static const unsigned short ucs_table_302A[] = { 01853 /* U+302A */ 0x302A, 01854 /* U+302B */ 0x302B, 01855 /* U+302C */ 0x302C, 01856 /* U+302D */ 0x302D, 01857 /* U+302E */ 0x302E, 01858 /* U+302F */ 0x302F, 01859 */ 0x302F, 01860 0x00, 01861 0x00, 01862 0x00, 01863 0x00, 01864 0x00, 01865 0x00, 01866 0x00, 01867 0x00, 01868 0x00, 01869 0x00, 01870 0x00, 01871 0x00, 01872 0x00, 01873 0x00, 01874 0x00, 01875 0x00, 01876 0x00, 01877 0x00, 01878 0x00, 01879 0x00, 01880 0x00, 01881 0x00, 01882 0x00, 01883 0x00, 01884 0x00, 01885 0x00, 01886 0x00, 01887 0x00, 01888 0x00, 01889 0x00, 01890 0x00, 01891 0x00, 01892 0x00, 01893 0x00, 01894 0x00, 01895 0x00, 01896 0x00, 01897 0x00, 01898 0x00, 01899 0x00, 01900 0x00, 01901 0x00, 01902 0x00, 01903 0x00, 01904 0x00, 01905 0x00, 01906 0x00, 01907 0x00, 01908 0x00, 01909 0x00, 01910 0x00, 01911 0x00, 01912 0x00,
```c
01913 0x00,
01914 0x00,
01915 0x00,
01916 0x00,
01917 0x00,
01918 0x00,
01919 0x00,
01920 0x00,
01921 0x00,
01922 0x00,
01923 0x00,
01924 0x00,
01925 0x00,
01926 0x00,
01927 0x00,
01928 0x00,
01929 0x00,
01930 0x00,
01931 0x00,
01932 0x00,
01933 0x00,
01934 0x00,
01935 0x00,
01936 0x00,
01937 0x00,
01938 0x00,
01939 0x00,
01940 0x00,
01941 0x00,
01942 0x00,
01943 0x00,
01944 0x00,
01945 0x00,
01946 0x00,
01947 0x00,
01948 0x00,
01949 0x00,
01950 0x00,
01951 0x00,
01952 0x00,
01953 0x00,
01954 0x00,
01955 0x00,
01956 0x00,
01957 0x00,
01958 0x00,
01959 0x00,
01960 0x00,
01961 0x00,
01962 0x00,
01963 0x00,
01964 0x00,
01965 /* U+3099 */ 0x309B,
01966 /* U+309A */ 0x309C,
01967 */
01968 static const unsigned short ucs_table_FB1E[] = {*
01969 /* U+FB1E */ 0xFB1E,
01970 */
01971 *
01972 static const unsigned short ucs_table_FE20[] = {*
01973 /* U+FE20 */ 0xFE20,
01974 /* U+FE21 */ 0xFE21,
01975 /* U+FE22 */ 0xFE22,
01976 /* U+FE23 */ 0xFE23,
01977 */
01978 */
13.256 symbol_.h

00001 /* symbol */
00002 static const char unicode_to_symbol_1b_0020[] = {*
00003 /* U+0020 */ 0x20,
00004 /* U+0021 */ 0x21,
00005 /* U+0023 */ 0x23,
00006 /* U+0025 */ 0x25,
00007 /* U+0026 */ 0x26,
00008 /* U+002B */ 0x2B,
00009 /* U+0028 */ 0x28,
00010 /* U+0029 */ 0x29,
00011 /* U+002C */ 0x2C,
```
00017 0x00,
00018 /* U+002E */ 0x2E,
00019 /* U+002F */ 0x2F,
00020 /* U+0030 */ 0x30,
00021 /* U+0031 */ 0x31,
00022 /* U+0032 */ 0x32,
00023 /* U+0033 */ 0x33,
00024 /* U+0034 */ 0x34,
00025 /* U+0035 */ 0x35,
00026 /* U+0036 */ 0x36,
00027 /* U+0037 */ 0x37,
00028 /* U+0038 */ 0x38,
00029 /* U+0039 */ 0x39,
00030 /* U+003A */ 0x3A,
00031 /* U+003B */ 0x3B,
00032 /* U+003C */ 0x3C,
00033 /* U+003D */ 0x3D,
00034 /* U+003E */ 0x3E,
00035 /* U+003F */ 0x3F,
00036 0x00,
00037 0x00,
00038 0x00,
00039 0x00,
00040 0x00,
00041 0x00,
00042 0x00,
00043 0x00,
00044 0x00,
00045 0x00,
00046 0x00,
00047 0x00,
00048 0x00,
00049 0x00,
00050 0x00,
00051 0x00,
00052 0x00,
00053 0x00,
00054 0x00,
00055 0x00,
00056 0x00,
00057 0x00,
00058 0x00,
00059 0x00,
00060 0x00,
00061 0x00,
00062 0x00,
00063 /* U+005B */ 0x5B,
00064 0x00,
00065 /* U+005C */ 0x5C,
00066 0x00,
00067 /* U+005D */ 0x5D,
00068 0x00,
00069 0x00,
00070 0x00,
00071 0x00,
00072 0x00,
00073 0x00,
00074 0x00,
00075 0x00,
00076 0x00,
00077 0x00,
00078 0x00,
00079 0x00,
00080 0x00,
00081 0x00,
00082 0x00,
00083 0x00,
00084 0x00,
00085 0x00,
00086 0x00,
00087 0x00,
00088 0x00,
00089 0x00,
00090 0x00,
00091 0x00,
00092 0x00,
00093 0x00,
00094 0x00,
00095 /* U+007B */ 0x7B,
00096 /* U+007C */ 0x7C,
00097 /* U+007D */ 0x7D,
00098 0x00,
00099 0x00,
00100 0x00,
00101 0x00,
00104 Ox0,
00105 Ox0,
00106 Ox0,
00107 Ox0,
00108 Ox0,
00109 Ox0,
00110 Ox0,
00111 Ox0,
00112 Ox0,
00113 Ox0,
00114 Ox0,
00115 Ox0,
00116 Ox0,
00117 Ox0,
00118 Ox0,
00119 Ox0,
00120 Ox0,
00121 Ox0,
00122 Ox0,
00123 Ox0,
00124 Ox0,
00125 Ox0,
00126 Ox0,
00127 Ox0,
00128 Ox0,
00129 Ox0,
00130 Ox0,
00131 Ox0,
00132 /* U+00A0 */ Ox20,
00133 Ox0,
00134 Ox0,
00135 Ox0,
00136 Ox0,
00137 Ox0,
00138 Ox0,
00139 Ox0,
00140 Ox0,
00141 Ox0,
00142 Ox0,
00143 Ox0,
00144 /* U+00AC */ Ox20,
00145 Ox0,
00146 Ox0,
00147 Ox0,
00148 /* U+00B0 */ Ox20,
00149 /* U+00B1 */ Ox20,
00150 Ox0,
00151 Ox0,
00152 Ox0,
00153 /* U+00D7 */ Ox20,
00154 Ox0,
00155 Ox0,
00156 Ox0,
00157 Ox0,
00158 Ox0,
00159 Ox0,
00160 Ox0,
00161 Ox0,
00162 Ox0,
00163 Ox0,
00164 Ox0,
00165 Ox0,
00166 Ox0,
00167 Ox0,
00168 Ox0,
00169 Ox0,
00170 Ox0,
00171 Ox0,
00172 Ox0,
00173 Ox0,
00174 Ox0,
00175 Ox0,
00176 Ox0,
00177 Ox0,
00178 Ox0,
00179 Ox0,
00180 Ox0,
00181 Ox0,
00182 Ox0,
00183 Ox0,
00184 Ox0,
00185 Ox0,
00186 Ox0,
00187 /* U+00D7 */ Ox20,
00188 Ox0,
00189 Ox0,
00190 Ox0,
static const char unicode_to_symbol_1b_0192[] = {
  /* U+0192 */ (char)0xA6,
  /* U+0391 */ 0x41,
  /* U+0392 */ 0x42,
  /* U+0393 */ 0x47,
  /* U+0394 */ 0x44,
  /* U+0395 */ 0x45,
  /* U+0396 */ 0x5A,
  /* U+0397 */ 0x48,
  /* U+0398 */ 0x51,
  /* U+0399 */ 0x49,
  /* U+039A */ 0x4B,
  /* U+039B */ 0x4C,
  /* U+039C */ 0x58,
  /* U+039D */ 0x4F,
  /* U+039E */ 0x50,
  /* U+039F */ 0x52,
  /* U+03A0 */ 0x53,
  /* U+03A1 */ 0x54,
  /* U+03A2 */ 0x55,
  /* U+03A3 */ 0x46,
  /* U+03A4 */ 0x43,
  /* U+03A5 */ 0x59,
  /* U+03A6 */ 0x57,
  /* U+03A7 */ 0x00,
  /* U+03A8 */ 0x00,
  /* U+03A9 */ 0x00,
  /* U+03AA */ 0x00,
  /* U+03AB */ 0x00,
  /* U+03AC */ 0x00,
  /* U+03AD */ 0x00,
  /* U+03AE */ 0x00,
  /* U+03AF */ 0x00,
  /* U+03B0 */ 0x00,
  /* U+03B1 */ 0x00,
  /* U+03B2 */ 0x00,
  /* U+03B3 */ 0x00,
  /* U+03B4 */ 0x00,
  /* U+03B5 */ 0x00,
  /* U+03B6 */ 0x00,
  /* U+03B7 */ 0x00,
  /* U+03B8 */ 0x00,
  /* U+03B9 */ 0x00,
  /* U+03BA */ 0x00,
  /* U+03BB */ 0x00,
  /* U+03BC */ 0x00,
  /* U+03BD */ 0x00,
  /* U+03BE */ 0x00,
  /* U+03BF */ 0x00,
  /* U+03C0 */ 0x00,
  /* U+03C1 */ 0x00,
  /* U+03C2 */ 0x00,
  /* U+03C3 */ 0x00,
};

static const char unicode_to_symbol_1b_0391[] = {
  /* U+0391 */ 0x41,
  /* U+0392 */ 0x42,
  /* U+0393 */ 0x47,
  /* U+0394 */ 0x44,
  /* U+0395 */ 0x45,
  /* U+0396 */ 0x5A,
  /* U+0397 */ 0x48,
  /* U+0398 */ 0x51,
  /* U+0399 */ 0x49,
  /* U+039A */ 0x4B,
  /* U+039B */ 0x4C,
  /* U+039C */ 0x58,
  /* U+039D */ 0x4F,
  /* U+039E */ 0x50,
  /* U+039F */ 0x52,
  /* U+03A0 */ 0x53,
  /* U+03A1 */ 0x54,
  /* U+03A2 */ 0x55,
  /* U+03A3 */ 0x46,
  /* U+03A4 */ 0x43,
  /* U+03A5 */ 0x59,
  /* U+03A6 */ 0x57,
  /* U+03A7 */ 0x00,
  /* U+03A8 */ 0x00,
  /* U+03A9 */ 0x00,
  /* U+03AA */ 0x00,
  /* U+03AB */ 0x00,
  /* U+03AC */ 0x00,
  /* U+03AD */ 0x00,
  /* U+03AE */ 0x00,
  /* U+03AF */ 0x00,
  /* U+03B0 */ 0x00,
  /* U+03B1 */ 0x00,
  /* U+03B2 */ 0x00,
  /* U+03B3 */ 0x00,
  /* U+03B4 */ 0x00,
  /* U+03B5 */ 0x00,
  /* U+03B6 */ 0x00,
  /* U+03B7 */ 0x00,
  /* U+03B8 */ 0x00,
  /* U+03B9 */ 0x00,
  /* U+03BA */ 0x00,
  /* U+03BB */ 0x00,
  /* U+03BC */ 0x00,
  /* U+03BD */ 0x00,
  /* U+03BE */ 0x00,
  /* U+03BF */ 0x00,
  /* U+03C0 */ 0x00,
  /* U+03C1 */ 0x00,
00365 0x00,
00366 0x00,
00367 0x00,
00368 0x00,
00369 0x00,
00370 0x00,
00371 0x00,
00372 0x00,
00373 0x00,
00374 0x00,
00375 0x00,
00376 0x00,
00377 0x00,
00378 0x00,
00379 0x00,
00380 0x00,
00381 0x00,
00382 0x00,
00383 0x00,
00384 0x00,
00385 0x00,
00386 0x00,
00387 0x00,
00388 0x00,
00389 0x00,
00390 0x00,
00391 0x00,
00392 0x00,
00393 0x00,
00394 0x00,
00395 0x00,
00396 0x00,
00397 0x00,
00398 0x00,
00399 0x00,
00400 0x00,
00401 0x00,
00402 0x00,
00403 0x00,
00404 0x00,
00405 0x00,
00406 0x00,
00407 0x00,
00408 0x00,
00409 0x00,
00410 0x00,
00411 0x00,
00412 0x00,
00413 0x00,
00414 0x00,
00415 0x00,
00416 0x00,
00417 0x00,
00418 0x00,
00419 0x00,
00420 0x00,
00421 0x00,
00422 0x00,
00423 0x00,
00424 0x00,
00425 0x00,
00426 0x00,
00427 0x00,
00428 0x00,
00429 0x00,
00430 0x00,
00431 0x00,
00432 0x00,
00433 0x00,
00434 0x00,
00435 0x00,
00436 0x00,
00437 0x00,
00438 /* U+20AC */ (char)0xA0,
00452 0x00, 00453 0x00, 00454 0x00, 00455 0x00, 00456 0x00, 00457 0x00, 00458 0x00, 00459 0x00, 00460 0x00, 00461 0x00, 00462 0x00, 00463 0x00, 00464 0x00, 00465 0x00, 00466 0x00, 00467 0x00, 00468 0x00, 00469 0x00, 00470 0x00, 00471 0x00, 00472 0x00, 00473 0x00, 00474 0x00, 00475 0x00, 00476 0x00, 00477 0x00, 00478 0x00, 00479 0x00, 00480 0x00, 00481 0x00, 00482 0x00, 00483 0x00, 00484 0x00, 00485 0x00, 00486 0x00, 00487 0x00, 00488 0x00, 00489 0x00, 00490 0x00, 00491 0x00, 00492 0x00, 00493 0x00, 00494 0x00, 00495 0x00, 00496 0x00, 00497 0x00, 00498 0x00, 00499 0x00, 00500 0x00, 00501 0x00, 00502 0x00, 00503 0x00, 00504 0x00, 00505 0x00, 00506 0x00, 00507 0x00, 00508 0x00, 00509 0x00, 00510 0x00, 00511 0x00, 00512 0x00, 00513 0x00, 00514 0x00, 00515 0x00, 00516 0x00, 00517 0x00, 00518 0x00, 00519 0x00, 00520 0x00, 00521 0x00, 00522 0x00, 00523 0x00, 00524 0x00, 00525 0x00, 00526 0x00, 00527 0x00, 00528 0x00, 00529 0x00, 00530 0x00, 00531 0x00, 00532 0x00, 00533 0x00, 00534 0x00, 00535 0x00, 00536 0x00, 00537 0x00, 00538 0x00,
00539 /* U+2111 */ (char)0xC1, 00540 0x00, 00541 0x00, 00542 0x00, 00543 0x00, 00544 0x00, 00545 0x00, 00546 /* U+2118 */ (char)0xC3, 00547 0x00, 00548 0x00, 00549 0x00, 00550 /* U+211C */ (char)0xC2, 00551 0x00, 00552 0x00, 00553 0x00, 00554 0x00, 00555 0x00, 00556 0x00, 00557 0x00, 00558 0x00, 00559 0x00, 00560 /* U+2126 */ 0x57, 00561 0x00, 00562 0x00, 00563 0x00, 00564 0x00, 00565 0x00, 00566 0x00, 00567 0x00, 00568 0x00, 00569 0x00, 00570 0x00, 00571 0x00, 00572 0x00, 00573 0x00, 00574 0x00, 00575 /* U+2135 */ (char)0xC0, 00576 0x00, 00577 0x00, 00578 0x00, 00579 0x00, 00580 0x00, 00581 0x00, 00582 0x00, 00583 0x00, 00584 0x00, 00585 0x00, 00586 0x00, 00587 0x00, 00588 0x00, 00589 0x00, 00590 0x00, 00591 0x00, 00592 0x00, 00593 0x00, 00594 0x00, 00595 0x00, 00596 0x00, 00597 0x00, 00598 0x00, 00599 0x00, 00600 0x00, 00601 0x00, 00602 0x00, 00603 0x00, 00604 0x00, 00605 0x00, 00606 0x00, 00607 0x00, 00608 0x00, 00609 0x00, 00610 0x00, 00611 0x00, 00612 0x00, 00613 0x00, 00614 0x00, 00615 0x00, 00616 0x00, 00617 0x00, 00618 0x00, 00619 0x00, 00620 0x00, 00621 0x00, 00622 0x00, 00623 0x00, 00624 0x00, 00625 0x00,
```
| 00626 | 0x00, |
| 00627 | 0x00, |
| 00628 | 0x00, |
| 00629 | 0x00, |
| 00630 | 0x00, |
| 00631 | 0x00, |
| 00632 | 0x00, |
| 00633 | 0x00, |
| 00634 | 0x00, |
| 00635 | 0x00, |
| 00636 | 0x00, |
| 00637 | 0x00, |
| 00638 | 0x00, |
| 00639 | 0x00, |
| 00640 | 0x00, |
| 00641 | 0x00, |
| 00642 | 0x00, |
| 00643 | 0x00, |
| 00644 | 0x00, |
| 00645 | 0x00, |
| 00646 | 0x00, |
| 00647 | 0x00, |
| 00648 | 0x00, |
| 00649 | 0x00, |
| 00650 | 0x00, |
| 00651 | 0x00, |
| 00652 | 0x00, |
| 00653 | 0x00, |
| 00654 | 0x00, |
| 00655 | 0x00, |
| 00656 | 0x00, |
| 00657 | 0x00, |
| 00658 | 0x00, |
| 00659 | 0x00, |
| 00660 | 0x00, |
| 00661 | 0x00, |
| 00662 | 0x00, |
| 00663 | 0x00, |
| 00664 | 0x00, |
| 00665 | 0x00, |
| 00666 | /* U+2190 */ (char)0xAC, |
| 00667 | /* U+2191 */ (char)0xAD, |
| 00668 | /* U+2192 */ (char)0xAE, |
| 00669 | /* U+2193 */ (char)0xAF, |
| 00670 | /* U+2194 */ (char)0xAB, |
| 00671 | 0x00, |
| 00672 | 0x00, |
| 00673 | 0x00, |
| 00674 | 0x00, |
| 00675 | 0x00, |
| 00676 | 0x00, |
| 00677 | 0x00, |
| 00678 | 0x00, |
| 00679 | 0x00, |
| 00680 | 0x00, |
| 00681 | 0x00, |
| 00682 | 0x00, |
| 00683 | 0x00, |
| 00684 | 0x00, |
| 00685 | 0x00, |
| 00686 | 0x00, |
| 00687 | 0x00, |
| 00688 | 0x00, |
| 00689 | 0x00, |
| 00690 | 0x00, |
| 00691 | 0x00, |
| 00692 | 0x00, |
| 00693 | 0x00, |
| 00694 | 0x00, |
| 00695 | 0x00, |
| 00696 | 0x00, |
| 00697 | 0x00, |
| 00698 | 0x00, |
| 00699 | 0x00, |
| 00700 | 0x00, |
| 00701 | 0x00, |
| 00702 | 0x00, |
| 00703 | /* U+2185 */ (char)0xBF, |
| 00704 | 0x00, |
| 00705 | 0x00, |
| 00706 | 0x00, |
| 00707 | 0x00, |
| 00708 | 0x00, |
| 00709 | 0x00, |
| 00710 | 0x00, |
| 00711 | 0x00, |
| 00712 | 0x00, |
```

Generated by Doxygen
00974 0x00,
00975 /* U+22C5 */ (char)0xD7,
00976 0x00,
00977 0x00,
00978 0x00,
00979 0x00,
00980 0x00,
00981 0x00,
00982 0x00,
00983 0x00,
00984 0x00,
00985 0x00,
00986 0x00,
00987 0x00,
00988 0x00,
00989 0x00,
00990 0x00,
00991 0x00,
00992 0x00,
00993 0x00,
00994 0x00,
00995 0x00,
00996 0x00,
00997 0x00,
00998 0x00,
00999 0x00,
01000 0x00,
01001 0x00,
01002 0x00,
01003 0x00,
01004 0x00,
01005 0x00,
01006 0x00,
01007 0x00,
01008 0x00,
01009 0x00,
01010 0x00,
01011 0x00,
01012 0x00,
01013 0x00,
01014 0x00,
01015 0x00,
01016 0x00,
01017 0x00,
01018 0x00,
01019 0x00,
01020 0x00,
01021 0x00,
01022 0x00,
01023 0x00,
01024 0x00,
01025 0x00,
01026 0x00,
01027 0x00,
01028 0x00,
01029 0x00,
01030 0x00,
01031 0x00,
01032 0x00,
01033 0x00,
01034 0x00,
01035 0x00,
01036 0x00,
01037 0x00,
01038 0x00,
01039 0x00,
01040 0x00,
01041 0x00,
01042 0x00,
01043 0x00,
01044 0x00,
01045 0x00,
01046 0x00,
01047 0x00,
01048 0x00,
01049 0x00,
01050 0x00,
01051 0x00,
01052 0x00,
01053 0x00,
01054 0x00,
01055 0x00,
01056 0x00,
01057 0x00,
01058 0x00,
01059 0x00,
01060 0x00,
13.257  imKStoUCS.c

```c

#include "Xlibint.h"
#include "Ximint.h"

static unsigned short const keysym_to_unicode_1a_1ff[] = {
    0x0104, 0x02d8, 0x0141, 0x0000, 0x013d, 0x015a, 0x0000, /* 0x01a0-0x01a7 */
    0x0000, 0x0160, 0x015e, 0x0164, 0x0179, 0x0000, 0x017d, 0x017b, /* 0x01a8-0x01af */
    0x0000, 0x0105, 0x02db, 0x0142, 0x0000, 0x013e, 0x015b, 0x02c7, /* 0x01b0-0x01b7 */
    0x0000, 0x0161, 0x015f, 0x0165, 0x017a, 0x02dd, 0x017e, 0x017c, /* 0x01b8-0x01bf */
    0x0154, 0x0000, 0x0000, 0x0102, 0x0000, 0x0139, 0x0106, 0x0000, /* 0x01c0-0x01c7 */
    0x010c, 0x0000, 0x0118, 0x0000, 0x011a, 0x0000, 0x010e, 0x0110, /* 0x01c8-0x01cf */
    0x0114, 0x0143, 0x0147, 0x0000, 0x0000, 0x0150, 0x0000, 0x0000, /* 0x01d0-0x01d7 */
    0x0158, 0x016e, 0x0000, 0x0170, 0x0000, 0x0162, 0x0000, 0x0000, /* 0x01d8-0x01df */
    0x0155, 0x0000, 0x0103, 0x0000, 0x013a, 0x0107, 0x0000, 0x0160 /* 0x01e0-0x01ee */
};

Generated by Doxygen
```
13.257 imKStoUCS.c

00191
00192
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277

static unsigned short keysym_to_unicode_12a1_12fe[]
0x1e02, 0x1e03, 0x0000, 0x0000, 0x0000,
0x1e80, 0x0000, 0x1e82, 0x1e0b, 0x1ef2, 0x0000,
0x1e1e, 0x1e1f, 0x0000, 0x0000, 0x1e40, 0x1e41,
0x1e81, 0x1e57, 0x1e83, 0x1e60, 0x1ef3, 0x1e84,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0174, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0175, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
};

2361

= {
0x1e0a,
0x0000,
0x0000,
0x1e85,
0x0000,
0x0000,
0x0000,
0x0176,
0x0000,
0x0000,
0x0000,
0x0177

0x0000,
0x0000,
0x1e56,
0x1e61,
0x0000,
0x0000,
0x1e6a,
0x0000,
0x0000,
0x0000,
0x1e6b,

/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*

0x12a0-0x12a7
0x12a8-0x12af
0x12b0-0x12b7
0x12b8-0x12bf
0x12c0-0x12c7
0x12c8-0x12cf
0x12d0-0x12d7
0x12d8-0x12df
0x12e0-0x12e7
0x12e8-0x12ef
0x12f0-0x12f7
0x12f0-0x12ff

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

static unsigned short const keysym_to_unicode_13bc_13be[] = {
0x0152, 0x0153, 0x0178
};

/* 0x13b8-0x13bf */

static unsigned short keysym_to_unicode_14a1_14ff[]
0x2741, 0x00a7, 0x0589, 0x0029, 0x0028,
0x2014, 0x002e, 0x055d, 0x002c, 0x2013, 0x058a,
0x055b, 0x055e, 0x0531, 0x0561, 0x0532, 0x0562,
0x0534, 0x0564, 0x0535, 0x0565, 0x0536, 0x0566,
0x0538, 0x0568, 0x0539, 0x0569, 0x053a, 0x056a,
0x053c, 0x056c, 0x053d, 0x056d, 0x053e, 0x056e,
0x0540, 0x0570, 0x0541, 0x0571, 0x0542, 0x0572,
0x0544, 0x0574, 0x0545, 0x0575, 0x0546, 0x0576,
0x0548, 0x0578, 0x0549, 0x0579, 0x054a, 0x057a,
0x054c, 0x057c, 0x054d, 0x057d, 0x054e, 0x057e,
0x0550, 0x0580, 0x0551, 0x0581, 0x0552, 0x0582,
0x0554, 0x0584, 0x0555, 0x0585, 0x0556, 0x0586,
};

= {
0x00bb,
0x2026,
0x0533,
0x0537,
0x053b,
0x053f,
0x0543,
0x0547,
0x054b,
0x054f,
0x0553,
0x2019,

0x00ab,
0x055c,
0x0563,
0x0567,
0x056b,
0x056f,
0x0573,
0x0577,
0x057b,
0x057f,
0x0583,
0x0027,

/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*

0x14a0-0x14a7
0x14a8-0x14af
0x14b0-0x14b7
0x14b8-0x14bf
0x14c0-0x14c7
0x14c8-0x14cf
0x14d0-0x14d7
0x14d8-0x14df
0x14e0-0x14e7
0x14e8-0x14ef
0x14f0-0x14f7
0x14f8-0x14ff

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

static unsigned short keysym_to_unicode_15d0_15f6[]
0x10d0, 0x10d1, 0x10d2, 0x10d3, 0x10d4, 0x10d5,
0x10d8, 0x10d9, 0x10da, 0x10db, 0x10dc, 0x10dd,
0x10e0, 0x10e1, 0x10e2, 0x10e3, 0x10e4, 0x10e5,
0x10e8, 0x10e9, 0x10ea, 0x10eb, 0x10ec, 0x10ed,
0x10f0, 0x10f1, 0x10f2, 0x10f3, 0x10f4, 0x10f5,
};

= {
0x10d6,
0x10de,
0x10e6,
0x10ee,
0x10f6

0x10d7,
0x10df,
0x10e7,
0x10ef,

/*
/*
/*
/*
/*

0x15d0-0x15d7
0x15d8-0x15df
0x15e0-0x15e7
0x15e8-0x15ef
0x15f0-0x15f7

*/
*/
*/
*/
*/

static unsigned short keysym_to_unicode_16a0_16f6[]
0x0000, 0x0000, 0xf0a2, 0x1e8a, 0x0000, 0xf0a5,
0xf0a8, 0x01b5, 0x01e6, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0xf0b2, 0x1e8b, 0x01d1, 0xf0b5,
0xf0b8, 0x01b6, 0x01e7, 0x0000, 0x0000, 0x01d2,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x1e36, 0xf0d2, 0xf0d3, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x1e37, 0xf0e2, 0xf0e3, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
};

= {
0x012c,
0x0000,
0x012d,
0x0000,
0x018f,
0x0000,
0x0000,
0x0000,
0x0000,
0x0000,
0x0259

0xf0a7,
0x019f,
0xf0b7,
0x0275,
0x0000,
0x0000,
0x0000,
0x0000,
0x0000,
0x0000,

/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*

0x16a0-0x16a7
0x16a8-0x16af
0x16b0-0x16b7
0x16b8-0x16bf
0x16c0-0x16c7
0x16c8-0x16cf
0x16d0-0x16d7
0x16d8-0x16df
0x16e0-0x16e7
0x16e8-0x16ef
0x16f0-0x16f6

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*

0x1ea0-0x1ea7
0x1ea8-0x1eaf
0x1eb0-0x1eb7
0x1eb8-0x1ebf
0x1ec0-0x1ec7
0x1ec8-0x1ecf
0x1ed0-0x1ed7
0x1ed8-0x1edf
0x1ee0-0x1ee7
0x1ee8-0x1eef
0x1ef0-0x1ef7
0x1ef8-0x1eff

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

static unsigned short const keysym_to_unicode_1e9f_1eff[] = {
0x0303,
0x1ea0, 0x1ea1, 0x1ea2, 0x1ea3, 0x1ea4, 0x1ea5, 0x1ea6, 0x1ea7,
0x1ea8, 0x1ea9, 0x1eaa, 0x1eab, 0x1eac, 0x1ead, 0x1eae, 0x1eaf,
0x1eb0, 0x1eb1, 0x1eb2, 0x1eb3, 0x1eb4, 0x1eb5, 0x1eb6, 0x1eb7,
0x1eb8, 0x1eb9, 0x1eba, 0x1ebb, 0x1ebc, 0x1ebd, 0x1ebe, 0x1ebf,
0x1ec0, 0x1ec1, 0x1ec2, 0x1ec3, 0x1ec4, 0x1ec5, 0x1ec6, 0x1ec7,
0x1ec8, 0x1ec9, 0x1eca, 0x1ecb, 0x1ecc, 0x1ecd, 0x1ece, 0x1ecf,
0x1ed0, 0x1ed1, 0x1ed2, 0x1ed3, 0x1ed4, 0x1ed5, 0x1ed6, 0x1ed7,
0x1ed8, 0x1ed9, 0x1eda, 0x1edb, 0x1edc, 0x1edd, 0x1ede, 0x1edf,
0x1ee0, 0x1ee1, 0x1ee2, 0x1ee3, 0x1ee4, 0x1ee5, 0x1ee6, 0x1ee7,
0x1ee8, 0x1ee9, 0x1eea, 0x1eeb, 0x1eec, 0x1eed, 0x1eee, 0x1eef,
0x1ef0, 0x1ef1, 0x0300, 0x0301, 0x1ef4, 0x1ef5, 0x1ef6, 0x1ef7,
0x1ef8, 0x1ef9, 0x01a0, 0x01a1, 0x01af, 0x01b0, 0x0309, 0x0323
};

static unsigned short const keysym_to_unicode_20a0_20ac[] = {
0x20a0, 0x20a1, 0x20a2, 0x20a3, 0x20a4, 0x20a5, 0x20a6, 0x20a7, /* 0x20a0-0x20a7 */
0x20a8, 0x20a9, 0x20aa, 0x20ab, 0x20ac
/* 0x20a8-0x20af */
};
static unsigned int
KeySymToUcs4(KeySym keysym)
{
/* ’Unicode keysym’ */
if ((keysym & 0xff000000) == 0x01000000)
return (keysym & 0x00ffffff);
if (keysym > 0 && keysym < 0x100)
return keysym;

Generated by Doxygen


else if (keysym > 0x1a0 && keysym < 0x200)
    return keysym_to_unicode_1a1_1ff[keysym - 0x1a1];
else if (keysym > 0x2a0 && keysym < 0x2ff)
    return keysym_to_unicode_2a1_2fe[keysym - 0x2a1];
else if (keysym > 0x3a1 && keysym < 0x3ff)
    return keysym_to_unicode_3a2_3fe[keysym - 0x3a2];
else if (keysym > 0x4a0 && keysym < 0x4e0)
    return keysym_to_unicode_4a1_4df[keysym - 0x4a1];
else if (keysym > 0x589 && keysym < 0x5ff)
    return keysym_to_unicode_590_5fe[keysym - 0x590];
else if (keysym > 0x67f && keysym < 0x700)
    return keysym_to_unicode_680_6ff[keysym - 0x680];
else if (keysym > 0x7a0 && keysym < 0x7fa)
    return keysym_to_unicode_7a1_7f9[keysym - 0x7a1];
else if (keysym > 0x8a3 && keysym < 0x8ff)
    return keysym_to_unicode_8a4_8fe[keysym - 0x8a4];
else if (keysym > 0x9de && keysym < 0x9f9)
    return keysym_to_unicode_9df_9f8[keysym - 0x9df];
else if (keysym > 0xaa0 && keysym < 0xaaf)
    return keysym_to_unicode_aa1_afe[keysym - 0xaa1];
else if (keysym > 0xcdf && keysym < 0xcf9)
    return keysym_to_unicode_cdf_cfa[keysym - 0xcdf];
else if (keysym > 0xda0 && keysym < 0xdf9)
    return keysym_to_unicode_da1_df9[keysym - 0xda1];
else if (keysym > 0xe9f && keysym < 0xff0)
    return keysym_to_unicode_ea0_ef0[keysym - 0xe9f];
else if (keysym > 0x12a0 && keysym < 0x12ff)
    return keysym_to_unicode_12a1_12fe[keysym - 0x12a1];
else if (keysym > 0x13bb && keysym < 0x13bf)
    return keysym_to_unicode_13bc_13be[keysym - 0x13bc];
else if (keysym > 0x14a0 && keysym < 0x1500)
    return keysym_to_unicode_14a1_1500[keysym - 0x14a1];
else if (keysym > 0x15cf && keysym < 0x15f7)
    return keysym_to_unicode_15d0_15f6[keysym - 0x15d0];
else if (keysym > 0x1e9e && keysym < 0x1eff)
    return keysym_to_unicode_1e9f_1eff[keysym - 0x1e9f];
else if (keysym > 0x209f && keysym < 0x20ad)
    return keysym_to_unicode_20a0_20ac[keysym - 0x20a0];
else
    return 0;
}

13.258  armscii_8.h

static const unsigned short armscii_8_2uni[96] = {
  0x00a0, 0xfffd, 0x0587, 0x0589, 0x0029, 0x0028, 0x00bb, 0x00ab,
  0x2014, 0x002e, 0x055d, 0x002c, 0x002d, 0x058a, 0x2026, 0x055c,
  0x055b, 0x055e, 0x0531, 0x0561, 0x0532, 0x0562, 0x0533, 0x0563,
  0x0534, 0x0564, 0x0555, 0x0556, 0x0557, 0x0558, 0x0559, 0x055a,
  0x0550, 0x0580, 0x0551, 0x0581, 0x0552, 0x0582, 0x0553, 0x0583,
  0x0554, 0x0584, 0x0555, 0x0585, 0x0556, 0x0586, 0x055a, 0xfffd,
};

static int
armscii_8_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
  unsigned char c = *s;
  if (c < 0xa0) {
    *pwc = (ucs4_t) c;
    return 1;
  }
  return 0;
}
return 1;
}

return RET_ILSEQ;

return RET_ILSEQ;

static const unsigned char armscii_8_page00[8] = {
    0xa5, 0xa4, 0x2a, 0x2b, 0xab, 0xac, 0xa9, 0x2f, /* 0x28-0x2f */
};

static const unsigned char armscii_8_page00_1[32] = {
    0xa0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
    0x00, 0x00, 0x00, 0xb0, 0x00, 0x00, 0x00, 0x00, / * 0xb0-0xb7 */
};

static const unsigned char armscii_8_page05[96] = {
    0x00, 0xb2, 0xb4, 0xb6, 0xb8, 0xba, 0xbc, 0xbe, /* 0x30-0x37 */
    0xc0, 0xc2, 0xc4, 0xc6, 0xc8, 0xca, 0xcc, 0xce, /* 0x38-0x3f */
    0xd0, 0xd2, 0xd4, 0xd6, 0xd8, 0xda, 0xdc, 0xde, /* 0x40-0x47 */
    0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, /* 0x48-0x4f */
    0xf0, 0xf2, 0xf4, 0xf6, 0xf8, 0xfa, 0xfc, 0x00, /* 0x50-0x57 */
    0x00, 0x00, 0xfe, 0xb0, 0xaf, 0xaa, 0xb1, 0x00, /* 0x58-0x5f */
    0x00, 0xb3, 0xb5, 0xb7, 0xb9, 0xbb, 0xbd, 0xbf, /* 0x60-0x67 */
    0xc1, 0xc3, 0xc5, 0xc7, 0xc9, 0xcb, 0xcc, 0xcf, /* 0x68-0x6f */
    0xd1, 0xd3, 0xd5, 0xd7, 0xd9, 0xdb, 0xdd, 0xdf, /* 0x70-0x77 */
    0xe1, 0xe3, 0xe5, 0xe7, 0xe9, 0xeb, 0xed, 0xef, /* 0x78-0x7f */
    0xf1, 0xf3, 0xf5, 0xf7, 0xf9, 0xfb, 0xfd, 0xff, /* 0x80-0x87 */
    0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c, 0x8e, /* 0x88-0x8f */

static int
armscii_8_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    unsigned char c = 0;
    if (wc < 0x0028) {  /* 0x00-0x00 */
        *r = wc;
        return 1;
    }
    else if (wc >= 0x0028 && wc < 0x00a0)  /* 0xa0-0x2f */
    c = armscii_8_page00[wc-0x0028];
    else if (wc >= 0x00a0 && wc < 0x00c0)  /* 0xa0-0x2f */
    c = armscii_8_page00_1[wc-0x00a0];
    else if (wc >= 0x0530 && wc < 0x0590)  /* 0xa0-0x2f */
    c = armscii_8_page05[wc-0x0530];
    else if (wc >= 0x2010 && wc < 0x2028)  /* 0xa0-0x2f */
    return RET_ILSEQ;
    *r = c;
    return 1;
}

return RET_ILSEQ;
}

/* ASCII */

static int
ascii_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = 0;
    if (wc < 0x0028) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x0028 && wc < 0x00a0)  /* 0xa0-0x2f */
    c = armscii_8_page00[wc-0x0028];
    else if (wc >= 0x00a0 && wc < 0x00c0)  /* 0xa0-0x2f */
    c = armscii_8_page00_1[wc-0x00a0];
    else if (wc >= 0x0530 && wc < 0x0590)  /* 0xa0-0x2f */
    c = armscii_8_page05[wc-0x0530];
    else if (wc >= 0x2010 && wc < 0x2028)  /* 0xa0-0x2f */
    return RET_ILSEQ;
    *r = c;
    return 1;
}

return RET_ILSEQ;

/* ASCII */

static int
ascii_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    if (wc < 0x0080) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x0080)  /* 0xa0-0x2f */
    c = armscii_8_page00[wc-0x0080];
    else if (wc >= 0x00a0 && wc < 0x00c0)  /* 0xa0-0x2f */
    c = armscii_8_page00_1[wc-0x00a0];
    else if (wc >= 0x0530 && wc < 0x0590)  /* 0xa0-0x2f */
    c = armscii_8_page05[wc-0x0530];
    else if (wc >= 0x2010 && wc < 0x2028)  /* 0xa0-0x2f */
    return RET_ILSEQ;
    *r = c;
    return 1;
}
static const unsigned short big5_2uni_pagea1[6121] = {
};

return RET_ILSEQ;

13.260 big5.h
static int
bg5_mbtowc (conv_t conv,ucs4_t *pwc, const unsigned char *s, int n)
{ 
unsigned char c1 = s[0];
if (c1 >= 0xa1 && c1 <= 0xff) { 
  if (n >= 2) { 
    unsigned char c2 = s[1];
    if ((c2 >= 0x40 && c2 < 0x7f) || (c2 >= 0xa1 && c2 < 0xff)) { 
      if (n >= 2) { 
        if (n >= 2) { 
          unsigned char c2 = s[1];
          if ((c2 >= 0x40 && c2 < 0x7f) || (c2 >= 0xa1 && c2 < 0xff)) { 
            unsigned int i = 157 * (c2 - 0xa1) + (c2 >= 0xa1 ? 0x52 : 0x40);
            unsigned short wc = 0xfffd;
            if (i < 6280) { 
              if (i < 6281) 
                wc = bg5_zwni_pageal[i];
            } else 
            return RET_ILSEQ;
          } 
          return RET_TOOFEM(0);
        } 
        return RET_ILSEQ;
      } 
      return RET_ILSEQ;
    } 
    if (wc != 0xfffd) { 
      *pwc = (ucs4_t) wc;
      return 2;
    } 
    return RET_ILSEQ;
  } 
  return RET_ILSEQ;
}

#define NEED_TOWC
static const unsigned short bg5_2charset[13703] = {
0x9c58, 0x9c50, 0x9c4a, 0x9c4d, 0x9c4b, 0x9c55, 0x9c59, 0x9c4c,
0x8c4e, 0x9c4f, 0x9d7, 0x9def, 0x9de3, 0x9de4, 0xd9f, 0xd9e8, 0xd9e4,
0x9de5, 0x9d0, 0x9de6, 0x9d0d, 0x9d0f, 0x9d0a, 0x9d0c, 0x9d02, 0x9d01, 0x9d02, 0x9d0f,
0x9b7, 0x9b6, 0x9b5, 0x9b4, 0x9b3, 0x9b2, 0x9b1, 0x9b0,
0x866a, 0x883e, 0x883d, 0x883f, 0x8b9e, 0x9f7b, 0x9f7a, 0x9f79, 0x571e, 0x7066, 0x7c6f, 0x883c, 0x8db2,
0x74db, 0x74e5, 0x77d5, 0x7938, 0x79b7, 0x79b6, 0x79b5, 0x9f06, 0x9f1c, 0x9f38, 0x9f37, 0x9f36, 0x9f35, 0x9f34, 0x9f33, 0x9f32, 0x9f31,
0x9f70, 0x9f6e, 0x9f6f, 0x9f71, 0x9f72, 0x9f73, 0x9f74, 0x9f75, 0x9f76, 0x9f77, 0x9f78, 0x9f79, 0x9f7a, 0x9f7c, 0x9f7b, 0x9f7d, 0x9f7e, 0x9f7f, 0x56d4, 0x652e, 0x65b8, 0x6b18,
0x9f46, 0x9f74, 0x9f75, 0x9f76, 0x9f77, 0x9f78, 0x9f79, 0x9f7a, 0x9f7c, 0x9f7b, 0x9f7d, 0x9f7e, 0x9f7f, 0x56d3, 0x652a, 0x652b, 0x6b1b, 0x6b19, 0x6b17, 0x6b1a, 0x7062, 0x7226, 0x722a, 0x72d8, 0x72d9,
0x7939, 0x7cf5, 0x7c5b, 0x7c5c, 0x7c6a, 0x9c5e, 0x9c60, 0x9e02, 0x9dfe, 0x9e07, 0x9e03, 0x9e06, 0x9e05,
0x8ea4, 0x8ea3, 0x846e, 0x946d, 0x946f, 0x9471, 0x9473, 0x9475,
0x9872, 0x959f, 0x9c68, 0x9c6c, 0x9c6d, 0xe06b, 0xe06d, 0xe06f,
0x80e0, 0x80e1, 0x80e2, 0x80e3, 0x80e4, 0x80e5, 0x80e6, 0x80e7,
0x7d7b, 0x7d7c, 0x7d7d, 0x7d7e, 0x7d7f, 0x7e98, 0x7e9b, 0x7e99, 0x81e0, 0x81e1, 0x81e2, 0x81e3,
0x9c79, 0x9c7a, 0x9c7b, 0x9c7c, 0x9c7d, 0x9c7e, 0x9c7f, 0x9e1d, 0x9e17, 0x9f48,
0x864c, 0x8ea8, 0x946b, 0x96b, 0x96c, 0x96d, 0x96e, 0x96f,
0x8e00, 0x8e01, 0x8e02, 0x8e03, 0x8e04, 0x8e05, 0x8e06, 0x8e07,
0x99ab, 0x9a64, 0x9a65, 0x9a66, 0x9a67, 0x9a68, 0x9a69, 0x9a6a,
0x999a, 0x999b, 0x999c, 0x999d, 0x999e, 0x999f, 0x99a0, 0x99a1,
0x9991, 0x9992, 0x9993, 0x9994, 0x9995, 0x9996, 0x9997, 0x9998,
0xa1f6, 0xa1f4, 0xa1f7, 0xa1f5, 0xa1f8, 0xa1f9, 0xa1fb, 0xa1fa,
0xa1ab, 0xa1b0, 0xa1c2, 0xa24a, 0xa1c1, 0xa24b, 0xa2ba, 0xa2bb,
0xa2bd, 0xa2bc, 0xa2be, 0xa2c2, 0xa2c1, 0xa2c0, 0xa1df, 0xa1fe, 0xa1f7, 0xa1f5, 0xa1f8, 0xa1f9, 0xa1fb, 0xa1fa,
0xa1d4, 0xa1db, 0xa1e8, 0xa1e7, 0xa1fd, 0xa1fc, 0xa1e4, 0xa1e5,
Generated by Doxygen

13.260 big5.h 2403
static const Summary16 big5_uni2indx_pagefa[] = {
  { 13458, 0xc6c3 }, { 13466, 0x5f6d }, { 13477, 0xff3d }, { 13490, 0x69ff },
  { 13502, 0xffcf }, { 13516, 0xfbf4 }, { 13528, 0xdcfb }, { 13540, 0x4ff7 },
  { 13552, 0x2000 }, { 13553, 0x1137 }, { 13560, 0x0015 },
  /* 0x9f00 */
};

static const Summary16 big5_uni2indx_pagefe[] = {
  { 13563, 0x3000 },
  /* 0xfa00 */
};

static int big5_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
  (void)conv;
  if (n >= 2) {
    const Summary16 *summary = NULL;
    if (wc < 0x0100) summary = &big5_uni2indx_page00[(wc»4)];
    else if (wc >= 0x0200 && wc < 0x0460) summary = &big5_uni2indx_page02[(wc»4)-0x20];
    else if (wc >= 0x2000 && wc < 0x22c0) summary = &big5_uni2indx_page20[(wc»4)-0x200];
    else if (wc >= 0x2400 && wc < 0x2650) summary = &big5_uni2indx_page24[(wc»4)-0x240];
    else if (wc >= 0x3000 && wc < 0x33e0) summary = &big5_uni2indx_page30[(wc»4)-0x300];
    else if (wc >= 0x4e00 && wc < 0x9fb0) summary = &big5_uni2indx_page4e[(wc»4)-0x4e0];
    else if (wc >= 0xfa00 && wc < 0xfa10) summary = &big5_uni2indx_pagefa[(wc»4)-0xfa0];
    else if (wc >= 0xfe00 && wc < 0xff70) summary = &big5_uni2indx_pagefe[(wc»4)-0xfe0];
    if (summary) {
      unsigned short used = summary->used;
      unsigned int i = wc & 0x0f;
      if (used & ((unsigned short) 1 « i)) {
        unsigned short c;
        / * Keep in 'used' only the bits 0..i-1. */
        used &= ((unsigned short) 1 « i) - 1;
        / * Add 'summary->indx' and the number of bits set in 'used'. */
        used = (used & 0x5555) + ((used & 0xaaaa) » 1);
        used = (used & 0x3333) + ((used & 0xc000) » 2);
        used = (used & 0x0f0f) + ((used & 0xf0f0) » 4);
        c = big5_2charset[summary->indx + used];
        r[0] = (c » 8); r[1] = (c & 0xff);
        return 2;
      }
    }
  }
  return RET_ILSEQ;
}

#endif /* NEED_TOMB */
Conversion (s1,s2) -> (c1,c2):

\[
t := 157 * (s1 - 0xA1) + (s2 < 0x80 ? s2 - 0x40 : s2 - 0x62)
\]
\[
c1 := (t div 94) + 0x21
\]
\[
c2 := (t mod 94) + 0x21
\]

Conversion (c1,c2) -> (s1,s2):

\[
t := 94 * (c1 - 0x21) + (c2 - 0x21)
\]
\[
s1 := (t div 157) + 0xA1
\]
\[
s2 := (t mod 157) <= 0x7E
\]

Conversion between BIG5 codes (s1,s2) and BIG5-1 codes (c1,c2):

Example. (s1,s2) = 0xC940, (c1,c2) = 0x2121.

\[
0xC9 <= s1 <= 0xF9, 0x40 <= s2 <= 0x7E \text{ or } 0xA1 <= s2 <= 0xFE,
\]
\[
0x21 <= c1 <= 0x72, 0x21 <= c2 <= 0x7E.
\]

Invariant:

\[
157 * (s1 - 0xC9) + (s2 < 0x80 ? s2 - 0x40 : s2 - 0x62) = 94 * (c1 - 0x21) + (c2 - 0x21)
\]

Conversion (s1,s2) -> (c1,c2):

\[
t := 157 * (s1 - 0xC9) + (s2 < 0x80 ? s2 - 0x40 : s2 - 0x62)
\]
\[
c1 := (t div 94) + 0xA1
\]
\[
t2 := t mod 157
\]
\[
s2 := (t2 < 0x3F ? t2 + 0x40 : t2 + 0x62)
\]

Conversion (c1,c2) -> (s1,s2):

\[
t := 94 * (c1 - 0x21) + (c2 - 0x21)
\]
\[
t2 := t mod 157
\]
\[
s1 := (t div 157) + 0xC9
\]
\[
s2 := (t2 < 0x3F ? t2 + 0x40 : t2 + 0x62)
\]

static int

big5_0_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)

{ }

if (c1 >= 0x21 && c1 <= 0x62) {
    if (n >= 2) {
        unsigned char c2 = s[1];
        if (c2 >= 0x21 && c2 <= 0x7e) {
            unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);
            if (i) {
                unsigned char buf[2];
                buf[0] = (i / 157) + 0xA1;
                buf[1] = i % 157;
                *(pwc) = (ucs4_t) buf;
                return big5_mbtowc(conv,pwc,buf,2);
            } else {
                return RET_ILSEQ;
            }
        } else {
            return RET_TOOFEW(0);
        }
    } else {
        return RET_ILSEQ;
    }
}

static int

big5_1_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)

{ }

if (c1 >= 0x21 && c1 <= 0x72) {
    if (n >= 2) {
        unsigned char c2 = s[1];
        if (c2 >= 0x21 && c2 <= 0x7e) {
            unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);
            if (i) {
                unsigned char buf[2];
                buf[0] = (i / 157) + 0xC9;
                buf[1] = i % 157;
                *(pwc) = (ucs4_t) buf;
                return big5_mbtowc(conv,pwc,buf,2);
            } else {
                return RET_ILSEQ;
            }
        } else {
            return RET_TOOFEW(0);
        }
    } else {
        return RET_ILSEQ;
    }
}

Generated by Doxygen
static int big5_0_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    if (n >= 2) {
        unsigned char buf[2];
        int ret = big5_wctomb(conv,buf,wc,2);
        if (ret != RET_ILSEQ) {
            unsigned char s1, s2;
            s1 = buf[0];
            s2 = buf[1];
            if (!((s1 >= 0xa1) && (s2 >= 0x40 && s2 <= 0x7e) || (s2 >= 0xa1 && s2 <= 0xfe))) abort();
            if ((s1 >= 0xc9) ||
                (s2 >= 0x40 && s2 <= 0x7e) ||
                (s2 >= 0x40 && s2 <= 0xfe))
                abort();
            if (s1 < 0xc9) {
                unsigned int t = 157 * (s1 - 0xa1) + s2 - (s2 < 0x80 ? 0x40 : 0x62);
                r[0] = (t / 94) + 0x21;
                r[1] = (t % 94) + 0x21;
                return 2;
            }
        }
        return RET_ILSEQ;
    }
    return RET_TOOSMALL;
}

static int big5_1_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    if (n >= 2) {
        unsigned char buf[2];
        int ret = big5_wctomb(conv,buf,wc,2);
        if (ret != RET_ILSEQ) {
            unsigned char s1, s2;
            s1 = buf[0];
            s2 = buf[1];
            if (!(s1 <= 0xf9)) abort();
            if (!(s2 >= 0x40 && s2 <= 0x7e) || (s2 >= 0xa1 && s2 <= 0xfe)) abort();
            if (s1 >= 0xc9) {
                unsigned int t = 157 * (s1 - 0xc9) + s2 - (s2 < 0x80 ? 0x40 : 0x62);
                r[0] = (t / 94) + 0x21;
                r[1] = (t % 94) + 0x21;
                return 2;
            }
        }
        return RET_ILSEQ;
    }
    return RET_TOOSMALL;
}
static int cpl133_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
  unsigned char c = *s;
  if (c < 0xa0) {
    *pwc = (ucs4_t) c;
    return 1;
  }
  else if (c < 0xe0) {
    unsigned short wc = cp1133_2uni_1[c-0xa0];
    if (wc != 0xfffd) {
      *pwc = (ucs4_t) wc;
      return 1;
    }
  }
  else if (c < 0xf0) {
    }
  else {
    unsigned short wc = cp1133_2uni_2[c-0xf0];
    if (wc != 0xfffd) {
      *pwc = (ucs4_t) wc;
      return 1;
    }
  }
  return RET_ILSEQ;
}

static const unsigned char cp1133_page00[16] = {
  0xa0, 0x00, 0xfc, 0x00, 0x00, 0x00, 0xfe, 0x00, /* 0xa0-0xa7 */
  0x00, 0x00, 0x00, 0x00, 0xfd, 0x00, 0x00, 0x00, /* 0xa8-0xaf */
};

static int cpl133_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
  unsigned char c = 0;
  if (wc < 0x00a0) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x00a0 && wc < 0x00b0) {
    c = cp1133_page00[wc-0x00a0];
  }
  else if (wc >= 0x0e80 && wc < 0x0ee0) {
    c = cp1133_page0e[wc-0x0e80];
  }
  else if (wc == 0x20ad) {
    c = 0xdf;
  }
  if (c != 0) {
    *r = c;
    return 1;
  };
  return RET_ILSEQ;
}

13.263 cp1251.h
static int cp1251_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c < 0x80) {
        *pwc = (ucs4_t) c;
        return 1;
    } else {
        unsigned short wc = cp1251_2uni[c-0x80];
        if (wc != 0xfffd) {
            *pwc = (ucs4_t) wc;
            return 1;
        }
    }
    return RET_ILSEQ;
}

static const unsigned char cp1251_page00[32] = {
    0xa0, 0x00, 0x00, 0x00, 0xa4, 0x00, 0xa6, /* 0xa0-0xa7 */
    0x00, 0xa9, 0x00, 0xab, 0xac, 0xad, 0xae, /* 0xa8-0xaf */
    0xb0, 0xb1, 0x00, 0x00, 0x00, 0xb5, 0xb6, 0xb7, /* 0xb0-0xb7 */
    0x00, 0x00, 0x00, 0xbb, 0x00, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x67 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x6f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
    0x00, 0x00, 0x90, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
    0x00, 0x88, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
    0xa5, 0xb4, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
};
```c
else if (wc >= 0x00a0 && wc < 0x00c0)
    c = cp1251_page00[wc-0x00a0];
else if (wc >= 0x0400 && wc < 0x0498)
    c = cp1251_page04[wc-0x0400];
else if (wc >= 0x2010 && wc < 0x2040)
    c = cp1251_page20[wc-0x2010];
else if (wc == 0x20ac)
    c = 0x88;
else if (wc == 0x2116)
    c = 0xb9;
else if (wc == 0x2122)
    c = 0x99;
if (c != 0) {
    *r = c;
    return 1;
}
return RET_ILSEQ;
#endif /* NEED_TOMB */

13.264  cp1255.h

static const unsigned short cp1255_2uni[128] = {
    0x20ac, 0xfffd, 0x201a, 0x0192, 0x201e, 0x2026, 0x2020, 0x2021,
    0x02c6, 0x2030, 0xfffd, 0x2039, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
    0xfffd, 0x2018, 0x2019, 0x201c, 0x201d, 0x2022, 0x2013, 0x2014,
    0x02dc, 0x2122, 0xfffd, 0x203a, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
    0x00a0, 0x00a1, 0x00a2, 0x00a3, 0x20aa, 0x00a5, 0x00a6, 0x00a7,
    0x00a8, 0x00a9, 0x00d7, 0x00ab, 0x00ac, 0x00ad, 0x00ae, 0x00af,
    0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7,
    0x00b8, 0x00b9, 0x00c7, 0x00ba, 0x00bc, 0x00bd, 0x00be, 0x00bf,
    0xc0 /* 0x80 */
```
13.265 cp1256.h

00046
00055 static const unsigned char cp1255_page02[32] = {
00056 0x00, 0x00, 0x00, 0x00, 0x00, 0x88, 0x00, 0x00, /* 0xc0-0xc7 */
00057 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
00058 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xd7 */
00059 0x00, 0x00, 0x00, 0x00, 0x98, 0x00, 0x00, 0x00, /* 0xd8-0xdf */
00060 }
00070
00071 static const unsigned char cp1255_page05[72] = {
00072 0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, /* 0xb0-0xb7 */
00073 0xc8, 0xc9, 0x00, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, /* 0xb8-0xbf */
00074 0xd0, 0xd1, 0xd2, 0xd3, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xc7 */
00075 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
00076 0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0xd0-0xd7 */
00077 0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* 0xd8-0xdf */
00078 0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* 0xe0-0xe7 */
00079 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe8-0xef */
00080 0x04, 0x05, 0xd6, 0xd7, 0xd8, 0x00, 0x00, 0x00, /* 0xf0-0xf7 */
00081 }
00082
00083 static int
00084 cp1255_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
00085 {
00086 unsigned char c = 0;
00087 if (wc < 0x0080) {
00088 *r = wc;
00089 return 1;
00090 }
00091 else if (wc >= 0x00a0 && wc < 0x00f8)
00092 c = cp1255_page00[wc-0x00a0];
00093 else if (wc == 0x0192)
00094 c = 0x83;
00095 else if (wc >= 0x02c0 && wc < 0x02e0)
00096 c = cp1255_page02[wc-0x02c0];
00097 else if (wc >= 0x05b0 && wc < 0x05f8)
00098 c = cp1255_page05[wc-0x05b0];
00099 else if (wc >= 0x2008 && wc < 0x2040)
00100 c = cp1255_page20[wc-0x2008];
00101 else if (wc == 0x20aa)
00102 c = 0xa4;
00103 else if (wc == 0x20ac)
00104 c = 0x80;
00105 else if (wc == 0x2122)
00106 c = 0x99;
00107 if (c != 0) {
00108 *r = c;
00109 return 1;
00110 }
00111 return RET_ILSEQ;
00112 }
00113

13.265 cp1256.h

00002
00003
00004 * CP1256
00005
00006
00007 static const unsigned short cp1256_2uni[128] = {
00008 / * 0x80 */
00009 0x20ac, 0x067e, 0x201a, 0x0192, 0x201e, 0x2026, 0x2020, 0x0211,
00010 0x0212, 0x0215, 0x0679, 0x2039, 0x0152, 0x088e, 0x0698, 0x0688,
00011 / * 0x90 */
00012 0x06a7, 0x06b0, 0x06b1, 0x06b2, 0x06b3, 0x06b4, 0x06b5, 0x06b6,
00013 / * 0xa0 */
00014 0x06ba, 0x06c0, 0x06c1, 0x06c2, 0x06c3, 0x06c4, 0x06c5, 0x06c6,
00015 / * 0xb0 */
00016 0x06d5, 0x06d6, 0x06d7, 0x06d8, 0x06d9, 0x06da, 0x06db, 0x06dc,
00017 / * 0xc0 */
00018 0x06e0, 0x06e1, 0x06e2, 0x06e3, 0x06e4, 0x06e5, 0x06e6, 0x06e7,
00019 / * 0xd0 */
00020 0x06f0, 0x06f1, 0x06f2, 0x06f3, 0x06f4, 0x06f5, 0x06f6, 0x06f7,
00021 / * 0xe0 */
00022 0x0700, 0x0701, 0x0702, 0x0703, 0x0704, 0x0705, 0x0706, 0x0707,
00023 / * 0xf0 */
00024 0x0710, 0x0711, 0x0712, 0x0713, 0x0714, 0x0715, 0x0716, 0x0717,
00025 / * 0x80 */
00026 0x0800, 0x0801, 0x0802, 0x0803, 0x0804, 0x0805, 0x0806, 0x0807,
if (wc < 0x0080) {
  *r = wc;
  return 1;
}
else if (wc >= 0x00a0 && wc < 0x0100)
  c = cp1256_page00[wc-0x00a0];
else if (wc >= 0x0150 && wc < 0x0198)
  c = cp1256_page01[wc-0x0150];
else if (wc == 0x02c6)
  c = 0x88;
else if (wc >= 0x0608 && wc < 0x06d8)
  c = cp1256_page06[wc-0x0608];
else if (wc >= 0x2008 && wc < 0x2040)
  c = cp1256_page20[wc-0x2008];
else if (wc == 0x20ac)
  c = 0x80;
else if (wc == 0x2122)
  c = 0x99;
if (c != 0) {
  *r = c;
  return 1;
}
return RET_ILSEQ;
}

/* Character encoding support for the Fast Light Tool Kit (FLTK).*/
/* Copyright 1998-2018 by Bill Spitzak and others.*/
/* This library is free software. Distribution and use rights are outlined in*/
/* the file "COPYING" which should have been included with this file. If this*/
/* file is missing or damaged, see the license at:*/
/* https://www.fltk.org/COPYING.php*/
/* Please see the following page on how to report bugs and issues:*/
/* https://www.fltk.org/bugs.php*/
/* PORTME: is this really needed? It's huge! */
/* not needed */

static int cp936ext_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) { return 0; }
#endif /* NEED_TOWC */

int cp936ext_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) { return 0; }
#endif /* NEED_TOMB */

static int cp936ext_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) { return 0; }
#endif /* NEED_TOMB */

static const ucs4_t cp936ext_2uni_page81[] = { /* Generated by Doxygen */

0x81 /* 0x81 */
0x4e02, 0x4e04, 0x4e05, 0x4e06, 0x4e0f, 0x4e12, 0x4e17, 0x4e1f,
0x4e20, 0x4e21, 0x4e23, 0x4e26, 0x4e29, 0x4e2e, 0x4e2f, 0x4e31,
0x4e33, 0x4e35, 0x4e37, 0x4e3c, 0x4e40, 0x4e41, 0x4e42, 0x4e44,
0x4e46, 0x4e4a, 0x4e50, 0x4e55, 0x4e57, 0x4e5a, 0x4e5b, 0x4e62,
0x4e63, 0x4e64, 0x4e65, 0x4e67, 0x4e68, 0x4e69, 0x4e6b, 0x4e6c,
0x4e6d, 0x4e6e, 0x4e70, 0x4e72, 0x4e74, 0x4e75, 0x4e76, 0x4e77,
0x4e78, 0x4e79, 0x4e7a, 0x4e7b, 0x4e7c, 0x4e7d, 0x4e7f, 0xe880,
0xe881, 0xe882, 0xe883, 0xe884, 0xe885, 0xe887, 0xe88a, 0xe890,
0xe896, 0xe897, 0xe899, 0xe8a0, 0xe8a1, 0xe8a2, 0xe8a3, 0xe8a4,
0xe8a5, 0xe8a6, 0xe8a7, 0xe8a8, 0xe8a9,
0xe8ac, 0xe8ae, 0xe8b0, 0xe8b1, 0xe8b4, 0xe8b5, 0xe8b7, 0xe8b8, 0xe8b9,
Generated by Doxygen
Generated by Doxygen
if (i < 23766) {
  unsigned short wc = 0xfffd;
  unsigned int i = 190 * (c1 - 0x81) + (c2 - (c2 >= 0x80 ? 0x41 : 0x40));
  if ((c2 >= 0x40 && c2 < 0x7f) || (c2 >= 0x80 && c2 < 0xff)) {
    unsigned char c2 = s[1];
    if (n >= 2) {
      if ((c1 >= 0x81 && c1 <= 0xfe)) {
        unsigned char c1 = s[0];
        ...
    }
  }
  ...
}

/* 0x00c */
...
/* 0x0e0 */
...
/* 0x0ed */
...
/* 0x0fe */
...
/* 0x100 */
...
/* 0x10f */
...
/* 0x118 */
...
/* 0x11e */
...
/* 0x120 */
...
/* 0x124 */
...
/* 0x12d */
...
/* 0x130 */
...
/* 0x134 */
...
/* 0x137 */
...
/* 0x13d */
...
/* 0x13f */
...
/* 0x140 */
...
/* 0x143 */
...
/* 0x145 */
...
/* 0x14a */
...
/* 0x14d */
...
/* 0x150 */
...
/* 0x153 */
...
/* 0x156 */
...
/* 0x157 */
...
/* 0x158 */
...
/* 0x15b */
...
/* 0x15e */
...
/* 0x161 */
...
/* 0x164 */
...
/* 0x165 */
...
/* 0x16a */
...
/* 0x16d */
...
/* 0x170 */
...
/* 0x173 */
...
/* 0x174 */
...
/* 0x175 */
...
/* 0x176 */
...
/* 0x177 */
...
/* 0x178 */
...
/* 0x179 */
...
/* 0x180 */
...
/* 0x181 */
...
/* 0x182 */
...
/* 0x183 */
...
/* 0x184 */
...
/* 0x185 */
...
/* 0x186 */
...
/* 0x187 */
...
/* 0x188 */
...
/* 0x189 */
...
/* 0x18a */
...
/* 0x18b */
...
/* 0x18c */
...
/* 0x18d */
...
/* 0x18e */
...
/* 0x18f */
...
/* 0x190 */
...
/* 0x191 */
...
/* 0x192 */
...
/* 0x193 */
...
/* 0x194 */
...
/* 0x195 */
...
/* 0x196 */
...
/* 0x197 */
...
/* 0x198 */
...
/* 0x199 */
...
/* 0x19a */
...
/* 0x19b */
...
/* 0x19c */
...
/* 0x19d */
...
/* 0x19e */
...
/* 0x19f */
...
/* 0x1a0 */
...
/* 0x1a1 */
...
/* 0x1a2 */
...
/* 0x1a3 */
...
/* 0x1a4 */
...
/* 0x1a5 */
...
/* 0x1a6 */
...
/* 0x1a7 */
...
/* 0x1a8 */
...
/* 0x1a9 */
...
/* 0x1aa */
...
/* 0x1ab */
...
/* 0x1ac */
...
/* 0x1ad */
...
/* 0x1ae */
...
/* 0x1af */
...
/* 0x1b0 */
...
/* 0x1b1 */
...
/* 0x1b2 */
...
/* 0x1b3 */
...
/* 0x1b4 */
...
/* 0x1b5 */
...
/* 0x1b6 */
...
/* 0x1b7 */
...
/* 0x1b8 */
...
/* 0x1b9 */
...
/* 0x1ba */
...
/* 0x1bb */
...
/* 0x1bc */
...
/* 0x1bd */
...
/* 0x1be */
...
/* 0x1bf */
...
/* 0x1c0 */
...
/* 0x1c1 */
...
/* 0x1c2 */
...
/* 0x1c3 */
...
/* 0x1c4 */
...
/* 0x1c5 */
...
/* 0x1c6 */
...
/* 0x1c7 */
...
/* 0x1c8 */
...
/* 0x1c9 */
...
/* 0x1ca */
...
/* 0x1cb */
...
/* 0x1cc */
...
/* 0x1cd */
...
/* 0x1ce */
...
/* 0x1cf */
...
/* 0x1d0 */
...
/* 0x1d1 */
...
/* 0x1d2 */
...
/* 0x1d3 */
...
/* 0x1d4 */
...
/* 0x1d5 */
...
/* 0x1d6 */
...
/* 0x1d7 */
...
/* 0x1d8 */
...
/* 0x1d9 */
...
/* 0x1da */
...
/* 0x1db */
...
/* 0x1dc */
...
/* 0x1dd */
...
/* 0x1de */
...
/* 0x1df */
...
/* 0x1e0 */
...
/* 0x1e1 */
...
/* 0x1e2 */
...
/* 0x1e3 */
...
/* 0x1e4 */
...
/* 0x1e5 */
...
/* 0x1e6 */
...
/* 0x1e7 */
...
/* 0x1e8 */
...
/* 0x1e9 */
...
/* 0x1ea */
...
/* 0x1eb */
...
/* 0x1ec */
...
/* 0x1ed */
...
/* 0x1ee */
...
/* 0x1ef */
...
/* 0x1f0 */
...
/* 0x1f1 */
...
/* 0x1f2 */
...
/* 0x1f3 */
...
/* 0x1f4 */
...
/* 0x1f5 */
...
/* 0x1f6 */
...
/* 0x1f7 */
...
/* 0x1f8 */
...
/* 0x1f9 */
...
/* 0x1fa */
...
/* 0x1fb */
...
/* 0x1fc */
...
/* 0x1fd */
...
/* 0x1fe */
...
/* 0x1ff */
...
if ((c1 >= 0x81 && c1 <= 0xff) && (c2 >= 0x80 && c2 < 0xff)) {
  unsigned int i = 190 * (c1 - 0x81) + (c2 - 0x80) + (c1 > 0x40 ? 0x41 : 0x40));
  unsigned short wc = cp936ext_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n);
  if (n >= 2) {
    if ((c1 >= 0x81 && c1 <= 0xfe)) {
      unsigned char c1 = s[0];
  ...
  }
static const unsigned short cp936ext_page04a0[232] = {
    ...
};

static const unsigned short cp936ext_page04a1[184] = {
    ...
};

static const unsigned short cp936ext_page04a2[64] = {
    ...
};

static const unsigned short cp936ext_page04a3[232] = {
    ...
};
static const unsigned short cp936ext_page09c0[20904] = {
.
.
};

static const unsigned short cp936ext_page0671[80] = {
.
.
};

static const unsigned short cp936ext_page0600[304] = {
.
.
};

static const unsigned short cp936ext_page04c0[72] = {
.
.
};

File Documentation
Generated by Doxygen
<table>
<thead>
<tr>
<th>Character Offset</th>
<th>Character</th>
<th>Character</th>
<th>Character</th>
<th>Character</th>
<th>Character</th>
<th>Character</th>
<th>Character</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>0x006a</td>
<td>0x006c</td>
<td>0x006d</td>
<td>0x006e</td>
<td>0x006f</td>
<td>0x0070</td>
<td>+0x08-0x0f</td>
<td></td>
</tr>
<tr>
<td>0x0001</td>
<td>0x0071</td>
<td>0x0072</td>
<td>0x0073</td>
<td>+0x06-0x07</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0002</td>
<td>0x0074</td>
<td>0x0075</td>
<td>0x0076</td>
<td>+0x06-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0003</td>
<td>0x0077</td>
<td>0x0078</td>
<td>0x0079</td>
<td>+0x06-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0004</td>
<td>0x007a</td>
<td>0x007b</td>
<td>0x007c</td>
<td>+0x07-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0005</td>
<td>0x007d</td>
<td>0x007e</td>
<td>0x007f</td>
<td>+0x07-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0006</td>
<td>0x0080</td>
<td>0x0081</td>
<td>0x0082</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0007</td>
<td>0x0084</td>
<td>0x0085</td>
<td>0x0086</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0008</td>
<td>0x0088</td>
<td>0x0089</td>
<td>0x008a</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0009</td>
<td>0x008c</td>
<td>0x008d</td>
<td>0x008e</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000a</td>
<td>0x0090</td>
<td>0x0091</td>
<td>0x0092</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000b</td>
<td>0x0094</td>
<td>0x0095</td>
<td>0x0096</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000c</td>
<td>0x0098</td>
<td>0x0099</td>
<td>0x009a</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000d</td>
<td>0x009c</td>
<td>0x009d</td>
<td>0x009e</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000e</td>
<td>0x00a0</td>
<td>0x00a1</td>
<td>0x00a2</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000f</td>
<td>0x00a4</td>
<td>0x00a5</td>
<td>0x00a6</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0010</td>
<td>0x00a8</td>
<td>0x00a9</td>
<td>0x00aa</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0011</td>
<td>0x00ab</td>
<td>0x00ac</td>
<td>0x00ad</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0012</td>
<td>0x00ae</td>
<td>0x00af</td>
<td>0x00b0</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0013</td>
<td>0x00b2</td>
<td>0x00b3</td>
<td>0x00b4</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0014</td>
<td>0x00b6</td>
<td>0x00b7</td>
<td>0x00b8</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0015</td>
<td>0x00ba</td>
<td>0x00bb</td>
<td>0x00bc</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0016</td>
<td>0x00be</td>
<td>0x00bf</td>
<td>0x00c0</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0017</td>
<td>0x00c2</td>
<td>0x00c3</td>
<td>0x00c4</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0018</td>
<td>0x00c6</td>
<td>0x00c7</td>
<td>0x00c8</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0019</td>
<td>0x00c9</td>
<td>0x00ca</td>
<td>0x00cb</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x001a</td>
<td>0x00cc</td>
<td>0x00cd</td>
<td>0x00ce</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x001b</td>
<td>0x00cf</td>
<td>0x00d0</td>
<td>0x00d1</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x001c</td>
<td>0x00d2</td>
<td>0x00d3</td>
<td>0x00d4</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x001d</td>
<td>0x00d6</td>
<td>0x00d7</td>
<td>0x00d8</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x001e</td>
<td>0x00d9</td>
<td>0x00da</td>
<td>0x00db</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x001f</td>
<td>0x00dc</td>
<td>0x00dd</td>
<td>0x00de</td>
<td>+0x08-0x0f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generated by Doxygen
static const unsigned short cp936ext_page1f41[40] = {
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0xd9df, 0xfd97, 0xfd98, 0xfd99, 0xfd9a, 0xfd9b, 0x0000, 0x0000,
    0xfd93, 0xc1fa, 0xb9a8, 0xede8, 0xfd94, 0xfd95, 0xfd96, 0xb9ea,
    0xfd8b, 0xfd8c, 0xfd8d, 0xfd8e, 0xfd8f, 0xfd90, 0xfd91, 0xfd92,
    0xf6b3, 0xfd86, 0xfd87, 0xf6b4, 0xc1e4, 0xf6b5, 0xf6b6, 0xf6b7,
    0xfd76, 0xfd77, 0xfd78, 0xfd79, 0xfd7a, 0xfd7b, 0xfd7c, 0xfd7d,
    0xfc50, 0xfc51, 0xfc52, 0xfc53, 0xbbc6, 0xfc54, 0xfc55, 0xfc56,
    0xf4ef, 0xfc4c, 0xfc4d, 0xc2e9, 0xfc4e, 0xf7e1, 0xf7e2, 0xfc4f,
    0xfb91, 0xfb92, 0xfb93, 0xfb94, 0xfb95, 0x7eae, 0xfb96, 0xfb97,
    0xf975, 0xf976, 0xf977, 0xf978, 0xf979, 0xf97a, 0xf97b, 0xf97c,
    0x9f00, 0x9f01, 0x9f02, 0x9f03, 0x9f04, 0x9f05, 0x9f06, 0x9f07,
    0x9f08, 0x9f09, 0x9f0a, 0x9f0b, 0x9f0c, 0x9f0d, 0x9f0e, 0x9f0f,
    0x9f10, 0x9f11, 0x9f12, 0x9f13, 0x9f14, 0x9f15, 0x9f16, 0x9f17,
    0x9f18, 0x9f19, 0x9f1a, 0x9f1b, 0x9f1c, 0x9f1d, 0x9f1e, 0x9f1f,
    0x9f20, 0x9f21, 0x9f22, 0x9f23, 0x9f24, 0x9f25, 0x9f26, 0x9f27,
    0x9f28, 0x9f29, 0x9f2a, 0x9f2b, 0x9f2c, 0x9f2d, 0x9f2e, 0x9f2f,
    0x9f30, 0x9f31, 0x9f32, 0x9f33, 0x9f34, 0x9f35, 0x9f36, 0x9f37,
    0x9f38, 0x9f39, 0x9f3a, 0x9f3b, 0x9f3c, 0x9f3d, 0x9f3e, 0x9f3f,
    0x9f40, 0x9f41, 0x9f42, 0x9f43, 0x9f44, 0x9f45, 0x9f46, 0x9f47,
    0x9f48, 0x9f49, 0x9f4a, 0x9f4b, 0x9f4c, 0x9f4d, 0x9f4e, 0x9f4f,
    0x9f50, 0x9f51, 0x9f52, 0x9f53, 0x9f54, 0x9f55, 0x9f56, 0x9f57,
    0x9f58, 0x9f59, 0x9f5a, 0x9f5b, 0x9f5c, 0x9f5d, 0x9f5e, 0x9f5f,
    0x9f60, 0x9f61, 0x9f62, 0x9f63, 0x9f64, 0x9f65, 0x9f66, 0x9f67,
    0x9f68, 0x9f69, 0x9f6a, 0x9f6b, 0x9f6c, 0x9f6d, 0x9f6e, 0x9f6f,
    0x9f70, 0x9f71, 0x9f72, 0x9f73, 0x9f74, 0x9f75, 0x9f76, 0x9f77,
    0x9f78, 0x9f79, 0x9f7a, 0x9f7b, 0x9f7c, 0x9f7d, 0x9f7e, 0x9f7f,
    0x9f80, 0x9f81, 0x9f82, 0x9f83, 0x9f84, 0x9f85, 0x9f86, 0x9f87,
    0x9f88, 0x9f89, 0x9f8a, 0x9f8b, 0x9f8c, 0x9f8d, 0x9f8e, 0x9f8f,
    0x9f90, 0x9f91, 0x9f92, 0x9f93, 0x9f94, 0x9f95, 0x9f96, 0x9f97,
    0x9f98, 0x9f99, 0x9f9a, 0x9f9b, 0x9f9c, 0x9f9d, 0x9f9e, 0x9f9f,
    0x9fa0, 0x9fa1, 0x9fa2, 0x9fa3, 0x9fa4, 0x9fa5, 0x9fa6, 0x9fa7,
    0x9fa8, 0x9fa9, 0x9faa, 0x9fab, 0x9fac, 0x9fad, 0x9fae, 0x9fac,
    0x9fb0, 0x9fb1, 0x9fb2, 0x9fb3, 0x9fb4, 0x9fb5, 0x9fb6, 0x9fb7,
    0x9fb8, 0x9fb9, 0x9fba, 0x9fbb, 0x9fbc, 0x9fbd, 0x9fbe, 0x9fbf,
static const unsigned short cp936ext_page1fc6[64] = {
    0xa955, 0xa6f2, 0x0000, 0xa6f4, 0xa6f5, 0xa6e0, 0xa6e1, 0xa6f0, /*0x30-0x37*/
    0xa6f1, 0xa6e2, 0xa6e3, 0xa6ee, 0xa6ef, 0xa6e6, 0xa6e7, 0xa6e4, /*0x38-0x3f*/
    0xa6e5, 0xa6e8, 0x0000, 0xa6e9, 0xa6e0, 0xa6e1, 0xa6f0, 0x0000, /*0x40-0x47*/
    0xa6f9, 0xa6fa, 0xa6fb, 0xa6fc, 0xa6fd, 0xa6fe, 0xa6ff, 0x0000, /*0x48-0x4f*/
    0xa700, 0xa701, 0xa702, 0xa703, 0xa704, 0xa705, 0xa706, 0xa707, /*0x50-0x57*/
    0xa708, 0xa709, 0xa70a, 0xa70b, 0xa70c, 0xa70d, 0xa70e, 0xa70f, /*0x58-0x5f*/
    0xa710, 0xa711, 0xa712, 0xa713, 0xa714, 0xa715, 0xa716, 0xa717, /*0x60-0x67*/
    0xa718, 0xa719, 0xa71a, 0xa71b, 0xa71c, 0xa71d, 0xa71e, 0x0000, /*0x68-0x6f*/
};

static const unsigned short cp936ext_page1fe0[96] = {
    0x0000, 0xa3a1, 0xa3a2, 0xa3a3, 0xa1e7, 0xa3a5, 0xa3a6, 0xa3a7, /*0x00-0x07*/
    0xa3a8, 0xa3a9, 0xa3aa, 0xa3ab, 0xa3ac, 0xa3ad, 0xa3ae, 0xa3af, /*0x08-0x0f*/
    0xa3b0, 0xa3b1, 0xa3b2, 0xa3b3, 0xa3b4, 0xa3b5, 0xa3b6, 0xa3b7, /*0x10-0x17*/
    0xa3b8, 0xa3b9, 0xa3ba, 0xa3bb, 0xa3bc, 0xa3bd, 0xa3be, 0xa3bf, /*0x18-0x1f*/
    0xa3c0, 0xa3c1, 0xa3c2, 0xa3c3, 0xa3c4, 0xa3c5, 0xa3c6, 0xa3c7, /*0x20-0x27*/
    0xa3c8, 0xa3c9, 0xa3ca, 0xa3cb, 0xa3cc, 0xa3cd, 0xa3ce, 0xa3cf, /*0x28-0x2f*/
    0xa3d0, 0xa3d1, 0xa3d2, 0xa3d3, 0xa3d4, 0xa3d5, 0xa3d6, 0xa3d7, /*0x30-0x37*/
    0xa3d8, 0xa3d9, 0xa3da, 0xa3db, 0xa3dc, 0xa3dd, 0xa3de, 0xa3df, /*0x38-0x3f*/
    0xa3e0, 0xa3e1, 0xa3e2, 0xa3e3, 0xa3e4, 0xa3e5, 0xa3e6, 0xa3e7, /*0x40-0x47*/
    0xa3e8, 0xa3e9, 0xa3ea, 0xa3eb, 0xa3ec, 0xa3ed, 0xa3ee, 0xa3ef, /*0x48-0x4f*/
    0xa3f0, 0xa3f1, 0xa3f2, 0xa3f3, 0xa3f4, 0xa3f5, 0xa3f6, 0xa3f7, /*0x50-0x57*/
    0xa3f8, 0xa3f9, 0xa3fa, 0xa3fb, 0xa3fc, 0xa3fd, 0xa1ab, 0x0000, /*0x58-0x5f*/
};

static const unsigned short cp936ext_page1ffc[8] = {
    0xa1e9, 0xa1ea, 0xa956, 0xa3fe, 0xa957, 0xa3a4, 0x0000, 0x0000, /*0xe0-0xe7*/
};

static int cp936ext_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
    if (n >= 2) {
        unsigned short c = 0;
        if (wc >= 0x00a0 && wc < 0x0170)
            c = cp936ext_page0014[wc-0x00a0];
        else if (wc >= 0x01c8 && wc < 0x01e0)
            c = cp936ext_page0039[wc-0x01c8];
        else if (wc >= 0x0250 && wc < 0x0268)
            c = cp936ext_page004a[wc-0x0250];
        else if (wc >= 0x02c0 && wc < 0x02e0)
            c = cp936ext_page0058[wc-0x02c0];
        else if (wc >= 0x0390 && wc < 0x03d0)
            c = cp936ext_page0072[wc-0x0390];
        else if (wc >= 0x0400 && wc < 0x0458)
            c = cp936ext_page0092[wc-0x0400];
        else if (wc >= 0x04b0 && wc < 0x0500)
            c = cp936ext_page0100[wc-0x04b0];
        else if (wc >= 0x0520 && wc < 0x0530)
            c = cp936ext_page0110[wc-0x0520];
        else if (wc >= 0x0540 && wc < 0x05e0)
            c = cp936ext_page0120[wc-0x0540];
        else if (wc >= 0x05f0 && wc < 0x0600)
            c = cp936ext_page0130[wc-0x05f0];
        else if (wc >= 0x0620 && wc < 0x0640)
            c = cp936ext_page0140[wc-0x0620];
        else if (wc >= 0x0650 && wc < 0x0660)
            c = cp936ext_page0150[wc-0x0650];
        else if (wc >= 0x0670 && wc < 0x0680)
            c = cp936ext_page0160[wc-0x0670];
        else if (wc >= 0x0690 && wc < 0x06a0)
            c = cp936ext_page0170[wc-0x0690];
        else if (wc >= 0x06b0 && wc < 0x06c0)
            c = cp936ext_page0180[wc-0x06b0];
        else if (wc >= 0x06d0 && wc < 0x06f0)
            c = cp936ext_page0190[wc-0x06d0];
        else if (wc >= 0x0700 && wc < 0x0710)
            c = cp936ext_page01a0[wc-0x0700];
        else if (wc >= 0x0720 && wc < 0x0730)
            c = cp936ext_page01b0[wc-0x0720];
        else if (wc >= 0x0740 && wc < 0x0750)
            c = cp936ext_page01c0[wc-0x0740];
        else if (wc >= 0x0760 && wc < 0x0770)
            c = cp936ext_page01d0[wc-0x0760];
        else if (wc >= 0x0780 && wc < 0x0790)
            c = cp936ext_page01e0[wc-0x0780];
        else if (wc >= 0x07a0 && wc < 0x07b0)
            c = cp936ext_page01f0[wc-0x07a0];
    }
    return c;
}
06236  c = cp936ext_page1ffc[wc-0xffe0];
06237  if (c != 0) {
06238      r[0] = (c >> 8); r[1] = (c & 0xff);
06239      return 2;
06240  }
06241  return RET_ILSEQ;
06242  }
06243  return RET_TOOSMALL;
06244  }
06245 #endif /* NEED_TOMB */
06246
06247 #endif /* CP936 */
06248
06249 #endif /* _WIN32 || __APPLE__ */ /* PORTME: Unicode stuff */
static int

static const unsigned short gb2312_2charset[7445] = {
  0x2168, 0x216c, 0x2127, 0x2163, 0x2140, 0x2141, 0x2824, 0x2822,
  0x2828, 0x283a, 0x282c, 0x282a, 0x2830, 0x2820, 0x284d, 0x285d,
  0x2838, 0x2836, 0x2837, 0x2839, 0x283b, 0x283c, 0x283d, 0x283e,
  0x283f, 0x2621, 0x2622, 0x2623, 0x2624, 0x2625, 0x2626, 0x2627,
  0x2628, 0x2629, 0x262a, 0x262b, 0x262c, 0x262d, 0x262e, 0x262f,
  0x2630, 0x2631, 0x2632, 0x2633, 0x2634, 0x2635, 0x2636, 0x2637,
  0x2638, 0x2639, 0x263a, 0x263b, 0x263c, 0x263d, 0x263e, 0x263f,
  0x2640, 0x2641, 0x2642, 0x2643, 0x2644, 0x2645, 0x2646, 0x2647,
  0x2648, 0x2649, 0x264a, 0x264b, 0x264c, 0x264d, 0x264e, 0x264f,
  0x2650, 0x2651, 0x2652, 0x2653, 0x2654, 0x2655, 0x2656, 0x2657,
  0x2658, 0x2659, 0x265a, 0x265b, 0x265c, 0x265d, 0x265e, 0x265f,
  0x2660, 0x2661, 0x2662, 0x2663, 0x2664, 0x2665, 0x2666, 0x2667,
  0x2668, 0x2669, 0x266a, 0x266b, 0x266c, 0x266d, 0x266e, 0x266f,
  0x2670, 0x2671, 0x2672, 0x2673, 0x2674, 0x2675, 0x2676, 0x2677,
  0x2678, 0x2679, 0x267a, 0x267b, 0x267c, 0x267d, 0x267e, 0x267f,
  0x2680, 0x2681, 0x2682, 0x2683, 0x2684, 0x2685, 0x2686, 0x2687,
  0x2688, 0x2689, 0x268a, 0x268b, 0x268c, 0x268d, 0x268e, 0x268f,
  0x2690, 0x2691, 0x2692, 0x2693, 0x2694, 0x2695, 0x2696, 0x2697,
  0x2698, 0x2699, 0x269a, 0x269b, 0x269c, 0x269d, 0x269e, 0x269f,
  0x26a0, 0x26a1, 0x26a2, 0x26a3, 0x26a4, 0x26a5, 0x26a6, 0x26a7,
  0x26a8, 0x26a9, 0x26aa, 0x26ab, 0x26ac, 0x26ad, 0x26ae, 0x26af,
  0x26b0, 0x26b1, 0x26b2, 0x26b3, 0x26b4, 0x26b5, 0x26b6, 0x26b7,
  0x26b8, 0x26b9, 0x26ba, 0x26bb, 0x26bc, 0x26bd, 0x26be, 0x26bf,
  0x26c0, 0x26c1, 0x26c2, 0x26c3, 0x26c4, 0x26c5, 0x26c6, 0x26c7,
  0x26c8, 0x26c9, 0x26ca, 0x26cb, 0x26cc, 0x26cd, 0x26ce, 0x26cf,
  0x26d0, 0x26d1, 0x26d2, 0x26d3, 0x26d4, 0x26d5, 0x26d6, 0x26d7,
  0x26d8, 0x26d9, 0x26da, 0x26db, 0x26dc, 0x26dd, 0x26de, 0x26df,
  0x26e0, 0x26e1, 0x26e2, 0x26e3, 0x26e4, 0x26e5, 0x26e6, 0x26e7,
  0x26e8, 0x26e9, 0x26ea, 0x26eb, 0x26ec, 0x26ed, 0x26ee, 0x26ef,
  0x26f0, 0x26f1, 0x26f2, 0x26f3, 0x26f4, 0x26f5, 0x26f6, 0x26f7,
  0x26f8, 0x26f9, 0x26fa, 0x26fb, 0x26fc, 0x26fd, 0x26fe, 0x26ff,
  0x2700, 0x2701, 0x2702, 0x2703, 0x2704, 0x2705, 0x2706, 0x2707,
  0x2708, 0x2709, 0x270a, 0x270b, 0x270c, 0x270d, 0x270e, 0x270f,
  0x2710, 0x2711, 0x2712, 0x2713, 0x2714, 0x2715, 0x2716, 0x2717,
  0x2718, 0x2719, 0x271a, 0x271b, 0x271c, 0x271d, 0x271e, 0x271f,
static const Summary16 gb2312_uni2indx_page9e[27] = {
  0x9c00, 0x9b00, 0x9a00, 0x9900, 0x9800, 0x9700, 0x9600, 0x9500, 0x9400, 0x9300, 0x9200, 0x9100, 0x9000, 0x8f00, 0x8e00, 0x8d00
};

Generated by Doxygen
static const Summary16 gb2312_uni2indx_pageff[15] = {
    7441, 0x0000, 7441, 0x0000, 7441, 0x0000, 7441, 0x0000, 7441, 0x0000, 7441, 0x0000, 7441, 0x0000, 7441, 0x0000, 7441, 0x0000, 7441, 0x0000, 7441, 0x0000, 7441, 0x0000,
};

static int gb2312_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
(void)conv;
if (n >= 2) {
    const Summary16 *summary = NULL;
    if (wc < 0x0460)
        summary = &gb2312_uni2indx_page00[(wc>>4)];
    else if (wc >= 0x2000 && wc < 0x2650)
        summary = &gb2312_uni2indx_page20[(wc>>4)-0x200];
    else if (wc >= 0x3000 && wc < 0x3230)
        summary = &gb2312_uni2indx_page30[(wc>>4)-0x300];
    else if (wc >= 0x4e00 && wc < 0x9cf0)
        summary = &gb2312_uni2indx_page4e[(wc>>4)-0x4e0];
    else if (wc >= 0x9e00 && wc < 0x9fb0)
        summary = &gb2312_uni2indx_page9e[(wc>>4)-0x9e0];
    else if (wc >= 0xff00 && wc < 0xfff0)
        summary = &gb2312_uni2indx_pageff[(wc>>4)-0xff0];
    if (summary) {
        unsigned short used = summary->used;
        unsigned int i = wc & 0x0f;
        if (used & ((unsigned short) 1 « i)) {
            unsigned short c;
            /* Keep in 'used' only the bits 0..i-1. */
            used &= ((unsigned short) 1 « i) - 1;
            /* Add 'summary->indx' and the number of bits set in 'used'. */
            used = (used & 0x5555) + ((used & 0xaaaa) » 1);
            used = (used & 0x3333) + ((used & 0xcccc) » 2);
            used = (used & 0x0f0f) + (used » 8);
            c = gb2312_2charset[summary->indx + used];
            r[0] = (c » 8); r[1] = (c & 0xff);
            return 2;
        }
    }
    return RET_ILSEQ;
}

return RET_TOOSMALL;
}

#define NEED_TOMB /*

13.268 georgian_academy.h

*/ #ifdef NEED_TOMB */

static const unsigned short georgian_academy_2uni[32] = {
    0x0080, 0x0081, 0x201a, 0x0192, 0x201e, 0x2026, 0x2020, 0x2021,
    0x02c6, 0x2030, 0x0160, 0x2039, 0x0152, 0x008d, 0x008e, 0x008f,
    0x0090, 0x2018, 0x2019, 0x201c, 0x201d, 0x2022, 0x2013, 0x2014,
    0x02dc, 0x2122, 0x0161, 0x023a, 0x0153, 0x009d, 0x009e, 0x0178,
};

static int georgian_academy_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c >= 0x80 && c < 0xa0)
        *pwc = (ucs4_t) georgian_academy_2uni[c-0x80];
    else if (c >= 0xc0 && c < 0xe7)
        *pwc = (ucs4_t) c + 0x1010;
    else
        *pwc = (ucs4_t) c;
    return 1;
}

#endif /* NEED_TOMB */
00030 0x80, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
00031 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
00032 0x00, 0x00, 0x00, 0x00, 0x00, 0x9d, 0x9e, 0x00, /* 0x88-0x8f */
00033 0x9f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
00034 );
00035 static const unsigned char georgian_academy_page01[72] = {
00036 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x50-0x57 */
00037 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x5f */
00038 0x9f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
00039 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
00040 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
00041 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
00042 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
00043 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
00044 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
00045 );
00046 static const unsigned char georgian_academy_page02[32] = {
00047 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x88, 0x00, /* 0xc0-0xc7 */
00048 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
00049 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xdf */
00050 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xdf */
00051 );
00052 static const unsigned char georgian_academy_page20[48] = {
00053 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x96, 0x97, /* 0x10-0x17 */
00054 0x91, 0x92, 0x82, 0x00, 0x93, 0x94, 0x84, 0x00, /* 0x18-0x1f */
00055 0x86, 0x87, 0x95, 0x00, 0x00, 0x00, 0x85, 0x00, /* 0x20-0x27 */
00056 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x3f */
00057 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x3f */
00058 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x3f */
00059 };,
00060 static int
00061 georgian_academy_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
00062 {
00063 unsigned char c = 0;
00064 if (wc < 0x0080) {
00065 *r = wc;
00066 return 1;
00067 }
00068 else if (wc >= 0x0080 && wc < 0x00a0)
00069 c = georgian_academy_page00[wc-0x0080];
00070 else if (wc >= 0x00a0 && wc < 0x00c0) || (wc >= 0x00e7 && wc < 0x0100)
00071 c = georgian_academy_page01[wc-0x0150];
00072 else if (wc >= 0x0150 && wc < 0x0198)
00073 c = georgian_academy_page02[wc-0x02c0];
00074 else if (wc >= 0x10d0 && wc < 0x10f7)
00075 c = wc-0x1010;
00076 else if (wc >= 0x2010 && wc < 0x2040)
00077 c = georgian_academy_page20[wc-0x2010];
00078 else if (wc >= 0x2122)
00079 c = 0x99;
00080 if (c != 0) {
00081 *r = c;
00082 return 1;
00083 }
00084 return RET_ILSEQ;
00085 }
00086

13.269 georgian_ps.h

00002
00003 /*
00004 * GEORGIAN-PS
00005 */
00006
00007 static const unsigned short georgian_ps_2uni_1[32] = {
00008 0x0080, 0x0081, 0x201a, 0x0192, 0x201e, 0x2026, 0x2020, /* 0x80 */
00009 0x10d0, 0x10d1, 0x10d2, 0x10d3, 0x10d4, 0x10d5, 0x10d6, 0x10d7, 0x10d8, 0x10d9, 0x10da, 0x10db, 0x10dc, 0x10dd, 0x10de, 0x10df, 0x10e0, 0x10e1, 0x10e2, 0x10e3, 0x10e4, 0x10e5, 0x10e6, 0x10e7, 0x10e8, 0x10e9, 0x10ea, 0x10eb, 0x10ec, 0x10ed, 0x10ee, 0x10ef, 0x10f0, 0x10f1, 0x10f2, 0x10f3, 0x10f4, 0x10f5, Generated by Doxygen
static int georgian_ps_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
    unsigned char c = *s;
    if (c >= 0x80 && c < 0xa0)
        *pwc = (ucs4_t) georgian_ps_2uni_1[c-0x80];
    else if (c >= 0xc0 && c < 0xe6)
        *pwc = (ucs4_t) georgian_ps_2uni_2[c-0xc0];
    else
        *pwc = (ucs4_t) c;
    return 1;
}

static const unsigned char georgian_ps_page00[32] = {
    0x80, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x8d, 0x8e, 0x8f, /* 0x88-0x8f */
    0x90, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x9d, 0x9e, 0x00, /* 0x98-0x9f */
};

static const unsigned char georgian_ps_page01[72] = {
    0x00, 0x00, 0x8c, 0x9c, 0x00, 0x00, 0x00, 0x00, /* 0x50-0x57 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x5f */
    0x8a, 0x9a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x67 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x6f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
    0x9f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
    0x00, 0x00, 0x00, 0x83, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
};

static const unsigned char georgian_ps_page02[32] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x88, 0x00, /* 0xc0-0xc7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xd7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x98, 0x00, 0x00, /* 0xd8-0xdf */
};

static const unsigned char georgian_ps_page10[40] = {
    0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc8, /* 0xd0-0xd7 */
    0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xcf, 0xd0, 0xd1, /* 0xd8-0xdf */
    0xd2, 0xd3, 0xd4, 0xd6, 0xd7, 0xda, 0xdb, 0xdc, /* 0xe0-0xe7 */
    0xdd, 0xde, 0xdf, 0xe0, 0xe1, 0xe3, 0xe4, 0xe5, /* 0xe8-0xef */
    0xe6, 0xe7, 0xea, 0xed, 0xe2, 0xe5, 0xe0, 0xe0, /* 0xf0-0xf7 */
};

static const unsigned char georgian_ps_page20[48] = {
    0x00, 0x00, 0x96, 0x97, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x17 */
    0x91, 0x92, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
    0x86, 0x87, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
    0x00, 0x8b, 0x9b, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x30-0x37 */
    0x00, 0x91, 0x92, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x38-0x3f */
};

static int georgian_ps_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
    unsigned char c = 0;
    if (wc < 0x080) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x0080 && wc < 0x0100)
        c = georgian_ps_page00[wc-0x0080];
    else if (wc >= 0x1000 && wc < 0x1010)
        c = georgian_ps_page10[wc-0x1000];
    else if (wc >= 0x2000 && wc < 0x2010)
        c = georgian_ps_page20[wc-0x2000];
    else
        return RET_ILSEQ;
}
```c
/* ISO-8859-1 */

static int iso8859_1_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
    unsigned char c = *s;
    *pwc = (ucs4_t) c;
    return 1;
}

static int iso8859_1_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
    if (wc < 0x0100) {
        *r = wc;
        return 1;
    } else
        return RET_ILSEQ;
}

/* ISO-8859-10 */

#ifdef NEED_TOWC
static const unsigned short iso8859_10_2uni[96] = {
    0x00a0, 0x0104, 0x0112, 0x0122, 0x012a, 0x0128, 0x0136, 0x00a7,
    0x013b, 0x0110, 0x0160, 0x0166, 0x017d, 0x00ad, 0x016a, 0x014a,
    0x00b0, 0x0105, 0x0113, 0x0123, 0x012b, 0x0129, 0x0137, 0x00b7,
    0x013c, 0x0111, 0x0161, 0x0167, 0x017e, 0x0205, 0x016b, 0x014b,
    0x00d0, 0x0145, 0x014c, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x012e,
    0x01d0, 0x0172, 0x00da, 0x00db, 0x00dc, 0x00dd, 0x00de, 0x00df,
    0x00e0, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x012f,
    0x01d0, 0x00e9, 0x0119, 0x00eb, 0x0117, 0x00ed, 0x00ee, 0x00ef,
    0x00f0, 0x0146, 0x014d, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x0169,
    0x00f0, 0x0173, 0x00fa, 0x00fb, 0x00fc, 0x00fd, 0x00fe, 0x0138,
};
#endif /* NEED_TOWC */

static int iso8859_10_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
    unsigned char c = *s;
    if (c < 0xa0)
        *pwc = (ucs4_t) c;
    else
        *pwc = (ucs4_t) iso8859_10_2uni[c-0xa0];
    return 1;
}

#else /* x86 */

#define NEED_TOWC

static const unsigned short iso8859_10_2uni[96] = {
    0x00a0, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x00a7, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x00b7, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x00d6, 0x00da, 0x00db, 0x00dc, 0x00dd, 0x00de, 0x00df, 0x00e0,
    0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x012f, 0x01d0,
    0x01e9, 0x0119, 0x01e0, 0x0117, 0x01ed, 0x01ee, 0x01ef, 0x00f0,
    0x0146, 0x014d, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x0169, 0x00f0,
    0x0173, 0x00fa, 0x00fb, 0x00fc, 0x00fd, 0x00fe, 0x0138,
};
#endif /* NEED_TOWC */

/* ISO-8859-10 */

```
static int iso8859_10_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
  (void)conv; (void)n;
  unsigned char c = 0;
  if (wc < 0x00a0) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x00a0 && wc < 0x0180) {
    c = iso8859_10_page00[wc-0x00a0];
  }
  else if (wc == 0x2015) {
    c = 0xbd;
  }
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}

static const unsigned short iso8859_11_2uni[96] = {
  0x00a0, 0x0e01, 0x0e02, 0x0e03, 0x0e04, 0x0e05, 0x0e06, 0x0e07,
  0x0e08, 0x0e09, 0x0e0a, 0x0e0b, 0x0e0c, 0x0e0d, 0x0e0e, 0x0e0f,
  0x0e10, 0x0e11, 0x0e12, 0x0e13, 0x0e14, 0x0e15, 0x0e16, 0x0e17,
  0x0e18, 0x0e19, 0x0e1a, 0x0e1b, 0x0e1c, 0x0e1d, 0x0e1e, 0x0e1f,
  0x0e20, 0x0e21, 0x0e22, 0x0e23, 0x0e24, 0x0e25, 0x0e26, 0x0e27,
  0x0e28, 0x0e29, 0x0e2a, 0x0e2b, 0x0e2c, 0x0e2d, 0x0e2e, 0x0e2f,
  0x0e30, 0x0e31, 0x0e32, 0x0e33, 0x0e34, 0x0e35, 0x0e36, 0x0e37,
  0x0e38, 0x0e39, 0x0e3a, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
  0x0e40, 0x0e41, 0x0e42, 0x0e43, 0x0e44, 0x0e45, 0x0e46, 0x0e47,
  0x0e48, 0x0e49, 0x0e4a, 0x0e4b, 0x0e4c, 0x0e4d, 0x0e4e, 0x0e4f,
  0x0e50, 0x0e51, 0x0e52, 0x0e53, 0x0e54, 0x0e55, 0x0e56, 0x0e57,
  0x0e58, 0x0e59, 0x0e5a, 0x0e5b, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
};

static int iso8859_11_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
  unsigned char c = *s;
  if (c < 0x80) {
    *pwc = (ucs4_t) c;
    return 1;
  }
  else if (c < 0xa0) {
    }
  else {
    unsigned short wc = iso8859_11_2uni[c-0xa0];
    if (wc != 0xfffd) {
      *pwc = (ucs4_t) wc;
      return 1;
    }
  }
}

13.272 iso8859_11.h
return RET_ILSEQ;
}
#endif

static const unsigned char iso8859_11_page0e[96] = {
  0x00, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, /* 0x00-0x07 */
  0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf, /* 0x08-0x0f */
  0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, /* 0x10-0x17 */
  0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf, /* 0x18-0x1f */
  0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, /* 0x20-0x27 */
  0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, /* 0x28-0x2f */
  0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, /* 0x30-0x37 */
  0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf, /* 0x38-0x3f */
  0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0x40-0x47 */
  0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* 0x48-0x4f */
  0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* 0x50-0x57 */
  0xf8, 0xf9, 0xfa, 0xfb, 0x00, 0x00, 0x00, 0xdf, /* 0x58-0x5f */
};

static int iso8859_11_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
  (void)conv; (void)n;
  unsigned char c = 0;
  if (wc < 0x0080 || wc == 0x00a0) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x0e00 && wc < 0x0e60)
    c = iso8859_11_page0e[wc-0x0e00];
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}
#endif

static int iso8859_13_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
  unsigned char c = 0;
  if (wc < 0x0080 || wc == 0x00a0) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x0e00 && wc < 0x0e60)
    c = iso8859_13_2uni[wc-0x0e00];
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}
#endif

static int iso8859_13_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
  unsigned char c = *s;
  if (c < 0xa0)
    *pwc = (ucs4_t) c;
  else
    *pwc = (ucs4_t) iso8859_13_2uni[c-0xa0];
  return 1;
}
#endif

static int iso8859_13_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
  unsigned char c = *s;
  if (c < 0xa0)
    *pwc = (ucs4_t) c;
  else
    *pwc = (ucs4_t) iso8859_13_2uni[c-0xa0];
  return 1;
}
#endif

static int iso8859_13_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
  unsigned char c = *s;
  if (c < 0xa0)
    *pwc = (ucs4_t) c;
  else
    *pwc = (ucs4_t) iso8859_13_2uni[c-0xa0];
  return 1;
}
static const unsigned char iso8859_13_page20[8] = {
00, 00, 00, 00, 00, 00, 00, 00, /* 0x00-0x0f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x10-0x1f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x20-0x2f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x30-0x3f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x40-0x4f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x50-0x5f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x60-0x6f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x70-0x7f */
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, /* 0x80-0xff */
};

static int iso8859_13_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
(void)conv; (void)n;
unsigned char c = 0;
if (wc < 0x00a0) {
    *r = wc;
    return 1;
}
else if (wc >= 0x00a0 && wc < 0x0180)
    c = iso8859_13_page00[wc-0x00a0];
else if (wc >= 0x2018 && wc < 0x2020)
    c = iso8859_13_page20[wc-0x2018];
if (c != 0) {
    *r = c;
    return 1;
}
return RET_ILSEQ;
}

static const unsigned char iso8859_13_page20[8] = {
00, 00, 00, 00, 00, 00, 00, 00, /* 0x00-0x0f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x10-0x1f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x20-0x2f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x30-0x3f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x40-0x4f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x50-0x5f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x60-0x6f */
00, 00, 00, 00, 00, 00, 00, 00, /* 0x70-0x7f */
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, /* 0x80-0xff */
};

static int iso8859_13_mbtowc (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
(void)conv; (void)n;
unsigned char c = 0;
if (wc < 0x00a0) {
    *r = wc;
    return 1;
}
else if (wc >= 0x00a0 && wc < 0x0180)
    c = iso8859_13_page00[wc-0x00a0];
else if (wc >= 0x2018 && wc < 0x2020)
    c = iso8859_13_page20[wc-0x2018];
if (c != 0) {
    *r = c;
    return 1;
}
return RET_ILSEQ;
}

13.274 iso8859_14.h

#define NEED_TOWC */
unsigned char c = *s;
if (c >= 0xa0)
  *pwc = (ucs4_t) iso8859_14_2uni[c-0xa0];
else
  *pwc = (ucs4_t) c;
return 1;
#endif /* NEED_TOWC */

#else
static const unsigned char iso8859_14_page00[96] = {
  0xa0, 0x00, 0x00, 0xa3, 0x00, 0x00, 0x00, 0xa7, /* 0xa0-0xa7 */
  0x00, 0xa9, 0x00, 0x00, 0x00, 0xad, 0xae, 0x00, /* 0xa8-0xaf */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xb6, 0x00, /* 0xb0-0xb7 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
  0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, /* 0xc0-0xc7 */
  0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, /* 0xc8-0xcf */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xd7 */
  0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xdf, 0x00, /* 0xd8-0xdf */
  0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0xe0-0xe7 */
  0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xef, 0x00, /* 0xe8-0xef */
  0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0x00, /* 0xf0-0xf7 */
  0x00, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0x00, 0x00, /* 0xf8-0xff */
};
static const unsigned char iso8859_14_page01_0[32] = {
  0x00, 0x00, 0xa4, 0xa5, 0x00, 0x00, 0x00, 0x00, /* 0x08-0x0f */
};
static const unsigned char iso8859_14_page01_1[16] = {
  0x00, 0x00, 0x00, 0x00, 0xd0, 0xf0, 0xde, 0xfe, /* 0x70-0x77 */
};
static const unsigned char iso8859_14_page1e_0[136] = {
  0x00, 0x00, 0xa1, 0xa2, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x07 */
  0x00, 0x00, 0xa6, 0xab, 0x00, 0x00, 0x00, 0x00, /* 0x08-0x0f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xb0, 0xb1, /* 0x10-0x1f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x28-0x2f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xb4, 0xb5, /* 0x30-0x37 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x38-0x3f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x40-0x47 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x48-0x4f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x50-0x57 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x5f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x67 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x6f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
};
static const unsigned char iso8859_14_page1e_1[8] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xf0-0xff */
};

static int iso8859_14_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  (void)conv; (void)n;
  unsigned char c = 0;
  if (wc < 0x00a0) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x00a0 && wc < 0x0100)
    c = iso8859_14_page00[wc-0xa0];
  else if (wc >= 0x0100 && wc < 0x0108)
    c = iso8859_14_page01_0[wc-0x0108];
  else if (wc >= 0x0108 && wc < 0x0110)
    c = iso8859_14_page01_1[wc-0x0108];
  return 1;
}
2528 File Documentation

13.275 iso8859_15.h

---

13.276 iso8859_16.h

---

Generated by Doxygen
static int iso8859_16_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c < 0xa0)
        *pwc = (ucs4_t) c;
    else
        *pwc = (ucs4_t) iso8859_16_2uni[c-0xa0];
    return 1;
}

static const unsigned char iso8859_16_page00[224] = {
    0xa0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xa7, /* 0xa0-0xa7 */
    0x00, 0xa9, 0x00, 0xab, 0x00, 0xad, 0x00, /* 0xa8-0xaf */
    0xb0, 0xb1, 0x00, 0x00, 0x00, 0x00, 0xb6, 0xb7, /* 0xb0-0xb7 */
    0x00, 0x00, 0xbb, 0xb8, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
    0xc0, 0xc1, 0xc2, 0x00, 0xc4, 0x00, 0xc6, 0xc7, /* 0xc0-0xc7 */
    0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, /* 0xc8-0xcf */
    0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, /* 0xd0-0xd7 */
    0x00, 0xd9, 0xda, 0xdb, 0xdc, 0x00, 0xdf, /* 0xd8-0xdf */
    0xe0, 0xe1, 0xe2, 0x00, 0xe4, 0x00, 0xe6, 0xe7, /* 0xe0-0xe7 */
    0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xef /* 0xe8-0xef */
};

static const unsigned char iso8859_16_page02[8] = {
    0xaa, 0xba, 0xde, 0xfe, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
};

static const unsigned char iso8859_16_page20[8] = {
    0x00, 0x00, 0xb5, 0xa5, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
};

static int iso8859_16_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    unsigned char c = 0;
    if (wc < 0x00a0) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x00a0 && wc < 0x0180)
        c = iso8859_16_page00[wc-0x00a0];
    else if (wc >= 0x0218 && wc < 0x0220)
        c = iso8859_16_page02[wc-0x0218];
    else if (wc >= 0x2018 && wc < 0x2020)
        c = iso8859_16_page20[wc-0x2018];
    else if (wc == 0x20ac)
        c = 0x44;
    else if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}

Generated by Doxygen
2530

File Documentation

13.277
00001
00002
00003
00004
00005
00006
00007
00008
00009
00010
00011
00012
00013
00014
00015
00016
00017
00018
00019
00020
00021
00022
00023
00024
00025
00026
00027
00028
00029
00030
00031
00032
00033
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
00077
00078
00079
00080
00081
00082
00083
00084
00085

iso8859_2.h

/*
* ISO-8859-2
*/
#ifdef NEED_TOWC
static const unsigned short iso8859_2_2uni[96] = {
/* 0xa0 */
0x00a0, 0x0104, 0x02d8, 0x0141, 0x00a4, 0x013d, 0x015a,
0x00a8, 0x0160, 0x015e, 0x0164, 0x0179, 0x00ad, 0x017d,
/* 0xb0 */
0x00b0, 0x0105, 0x02db, 0x0142, 0x00b4, 0x013e, 0x015b,
0x00b8, 0x0161, 0x015f, 0x0165, 0x017a, 0x02dd, 0x017e,
/* 0xc0 */
0x0154, 0x00c1, 0x00c2, 0x0102, 0x00c4, 0x0139, 0x0106,
0x010c, 0x00c9, 0x0118, 0x00cb, 0x011a, 0x00cd, 0x00ce,
/* 0xd0 */
0x0110, 0x0143, 0x0147, 0x00d3, 0x00d4, 0x0150, 0x00d6,
0x0158, 0x016e, 0x00da, 0x0170, 0x00dc, 0x00dd, 0x0162,
/* 0xe0 */
0x0155, 0x00e1, 0x00e2, 0x0103, 0x00e4, 0x013a, 0x0107,
0x010d, 0x00e9, 0x0119, 0x00eb, 0x011b, 0x00ed, 0x00ee,
/* 0xf0 */
0x0111, 0x0144, 0x0148, 0x00f3, 0x00f4, 0x0151, 0x00f6,
0x0159, 0x016f, 0x00fa, 0x0171, 0x00fc, 0x00fd, 0x0163,
};

0x00a7,
0x017b,
0x02c7,
0x017c,
0x00c7,
0x010e,
0x00d7,
0x00df,
0x00e7,
0x010f,
0x00f7,
0x02d9,

static int
iso8859_2_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
unsigned char c = *s;
if (c < 0xa0)
*pwc = (ucs4_t) c;
else
*pwc = (ucs4_t) iso8859_2_2uni[c-0xa0];
return 1;
}
#endif /* NEED_TOWC */
#ifdef NEED_TOMB
static const unsigned char iso8859_2_page00[224] = {
0xa0, 0x00, 0x00, 0x00, 0xa4, 0x00, 0x00, 0xa7, /*
0xa8, 0x00, 0x00, 0x00, 0x00, 0xad, 0x00, 0x00, /*
0xb0, 0x00, 0x00, 0x00, 0xb4, 0x00, 0x00, 0x00, /*
0xb8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0x00, 0xc1, 0xc2, 0x00, 0xc4, 0x00, 0x00, 0xc7, /*
0x00, 0xc9, 0x00, 0xcb, 0x00, 0xcd, 0xce, 0x00, /*
0x00, 0x00, 0x00, 0xd3, 0xd4, 0x00, 0xd6, 0xd7, /*
0x00, 0x00, 0xda, 0x00, 0xdc, 0xdd, 0x00, 0xdf, /*
0x00, 0xe1, 0xe2, 0x00, 0xe4, 0x00, 0x00, 0xe7, /*
0x00, 0xe9, 0x00, 0xeb, 0x00, 0xed, 0xee, 0x00, /*
0x00, 0x00, 0x00, 0xf3, 0xf4, 0x00, 0xf6, 0xf7, /*
0x00, 0x00, 0xfa, 0x00, 0xfc, 0xfd, 0x00, 0x00, /*
/* 0x0100 */
0x00, 0x00, 0xc3, 0xe3, 0xa1, 0xb1, 0xc6, 0xe6, /*
0x00, 0x00, 0x00, 0x00, 0xc8, 0xe8, 0xcf, 0xef, /*
0xd0, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0xca, 0xea, 0xcc, 0xec, 0x00, 0x00, 0x00, 0x00, /*
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0x00, 0xc5, 0xe5, 0x00, 0x00, 0xa5, 0xb5, 0x00, /*
0x00, 0xa3, 0xb3, 0xd1, 0xf1, 0x00, 0x00, 0xd2, /*
0xf2, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0xd5, 0xf5, 0x00, 0x00, 0xc0, 0xe0, 0x00, 0x00, /*
0xd8, 0xf8, 0xa6, 0xb6, 0x00, 0x00, 0xaa, 0xba, /*
0xa9, 0xb9, 0xde, 0xfe, 0xab, 0xbb, 0x00, 0x00, /*
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xd9, 0xf9, /*
0xdb, 0xfb, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0x00, 0xac, 0xbc, 0xaf, 0xbf, 0xae, 0xbe, 0x00, /*
};
static const unsigned char iso8859_2_page02[32] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xb7, /*
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0xa2, 0xff, 0x00, 0xb2, 0x00, 0xbd, 0x00, 0x00, /*
};

0xa0-0xa7
0xa8-0xaf
0xb0-0xb7
0xb8-0xbf
0xc0-0xc7
0xc8-0xcf
0xd0-0xd7
0xd8-0xdf
0xe0-0xe7
0xe8-0xef
0xf0-0xf7
0xf8-0xff

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

0x00-0x07
0x08-0x0f
0x10-0x17
0x18-0x1f
0x20-0x27
0x28-0x2f
0x30-0x37
0x38-0x3f
0x40-0x47
0x48-0x4f
0x50-0x57
0x58-0x5f
0x60-0x67
0x68-0x6f
0x70-0x77
0x78-0x7f

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

0xc0-0xc7
0xc8-0xcf
0xd0-0xd7
0xd8-0xdf

*/
*/
*/
*/

/*
static int
iso8859_2_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
unsigned char c = 0;
if (wc < 0x00a0) {

Generated by Doxygen


...
43.279 iso8859_4.h

00002 00003 #ifdef NEED_TOWC
00004 static const unsigned short iso8859_4_2uni[96] = {
00005 / * 0xa0 */
00006 0x00a0, 0x0104, 0x0138, 0x0156, 0x00a4, 0x0128, 0x013b, 0x00a7,
00007 / * 0xb0 */
00008 0x00b0, 0x0105, 0x02db, 0x0157, 0x00b4, 0x0129, 0x013c, 0x02c7,
00009 / * 0xc0 */
00010 0x0100, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x012a,
00011 / * 0xd0 */
00012 0x0110, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
00013 / * 0xe0 */
00014 0x0101, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7,
00015 / * 0xf0 */
00016 0x0111, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x00f7,
00017 0x00f8, 0x0173, 0x00fa, 0x00fb, 0x00fc, 0x0169, 0x016a, 0x02d9,
00018 0x02da;}
00019 static int
00020 iso8859_4_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
00021 {
00022 unsigned char c = *s;
00023 if (c < 0xa0)
00024 *pwc = (ucs4_t) c;
00025 else
00026 *pwc = (ucs4_t) iso8859_4_2uni[c-0xa0];
00027 return 1;
00028 }
00029 #endif /* NEED_TOWC */
`iso8859_5.h`

```c
00045 0xb0, 0x00, 0x00, 0x00, 0xb4, 0x00, 0x00, 0x00, /* 0xb0-0xb7 */
00046 0xb8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
00047 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0x00, /* 0xc0-0xc7 */
00048 0xc9, 0x00, 0xcb, 0xc0, 0xce, 0x00, /* 0xc8-0xcf */
00049 0xd4, 0xd5, 0xd6, 0xd7, /* 0xd0-0xd7 */
00050 0xd8, 0x00, 0xda, 0xdb, 0xdc, 0x00, 0x00, 0xdf, /* 0xd8-0xdf */
00051 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0x00, /* 0xe0-0xe7 */
00052 0xe9, 0x00, 0xeb, 0xe0, 0xed, 0xee, 0x00, /* 0xe8-0xef */
00053 0xf4, 0xf5, 0xf6, 0xf7, /* 0xf0-0xf7 */
00054 0xf8, 0x00, 0xfa, 0xfb, 0xfc, 0x00, 0x00, 0xff, /* 0xf8-0xff */
00055 */ 0x0100 */
00056 0xc0, 0xe0, 0x00, 0x00, 0xa1, 0xb1, 0x00, 0x00, /* 0x00-0x07 */
00057 0x00, 0x00, 0x00, 0x00, 0xc8, 0xe8, 0x00, 0x00, /* 0x08-0x0f */
00058 0xd0, 0xf0, 0xaa, 0xba, 0x00, 0x00, 0xc8, 0xe8, /* 0x10-0x17 */
00059 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f, 0x00, 0x00, /* 0x18-0x1f */
00060 0x2, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
00061 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x28-0x2f */
00062 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x30-0x37 */
00063 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x38-0x3f */
00064 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x40-0x47 */
00065 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x48-0x4f */
00066 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x50-0x57 */
00067 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x5f */
00068 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x67 */
00069 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x6f */
00070 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
00071 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
00072 };
00073 static const unsigned char iso8859_4_page02[32] = {
00074 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xb7, /* 0xc0-0xc7 */
00075 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
00076 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xdf */
00077 0x00, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xef */
00078 
00079 
00080 static int
00081 iso8859_4_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
00082 
00083 { (void)conv; (void)n;
00084 unsigned char c = 0;
00085 if (wc < 0x00a0) {
00086   *r = wc;
00087   return 1;
00088 
00089 } else if (wc >= 0x00a0 && wc < 0x0180)
00090   c = iso8859_4_page00[wc-0x00a0];
00091 else if (wc >= 0x02c0 && wc < 0x02e0)
00092   c = iso8859_4_page02[wc-0x02c0];
00093 if (c != 0) {
00094     *r = c;
00095     return 1;
00096   } else { return RET_ILSEQ; }
00097 }
00098 
00099 #endif /* NEED_TOMB */
```

### iso8859_5.h

```c
00101 */
00102 static int
00103 iso8859_5_wctomb (conv_t conv, unsigned char *r, wc4_t wc, int n)
00104 
00105 
00106 static int
00107 iso8859_5_wctomb (conv_t conv, unsigned char *r, wc4_t wc, int n)
00108 
00109 
00110 // Generated by Doxygen
```

13.280 iso8859_5.h

```c
00111 #ifdef NEED_TOWC
00112 static const unsigned short iso8859_5_2uni[96] = {
00113 0x00a0, 0x00a1, 0x00a2, 0x00a3, 0x00a4, 0x00a5, 0x00a6, 0x00a7, /* 0x00a0-0x00bf */
00114 0x00c0, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7, /* 0x00c0-0x00df */
00115 0x00e0, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7, /* 0x00e0-0x00ff */
00116 0x00f0, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x00f7, /* 0x00f0-0x00ff */
00117 0x00f8, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x00fd, 0x00fe, 0x00ff, /* 0x00f8-0x00ff */
00118 #endif /* NEED_TOWC */
```
static int iso8859_5_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
    unsigned char c = *s;
    if (c < 0xa0)
        *pwc = (ucs4_t) c;
    else
        *pwc = (ucs4_t) iso8859_5_2uni[c-0xa0];
    return 1;
}

static const unsigned char iso8859_5_page00[16] = {
    0xa0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfd, /* 0xa0-0xa7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0xad, 0x00, 0x00, /* 0xa8-0xaf */
};

static int iso8859_5_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
    (void)conv; (void)n;
    unsigned char c = 0;
    if (wc < 0x00a0) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x00a0 && wc < 0xb0)
        c = iso8859_5_page00[wc-0x00a0];
    else if (wc >= 0x0400 && wc < 0x0460)
        c = iso8859_5_page04[wc-0x0400];
    else if (wc == 0x2116)
        c = 0xf0;
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}

static const unsigned short iso8859_6_2uni[96] = {
    0x00a0, 0xfffd, 0xfffd, 0xfffd, 0x00a4, 0xfffd, 0xfffd, 0xfffd,
    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0x060c, 0x00ad, 0xfffd, 0xfffd,
    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
    0x0630, 0x0631, 0x0632, 0x0633, 0x0634, 0x0635, 0x0636, 0x0637,
    0x0638, 0x0639, 0x063a, 0x063b, 0x063c, 0x063d, 0x063e, 0x063f,
    0x0640, 0x0641, 0x0642, 0x0643, 0x0644, 0x0645, 0x0646, 0x0647,
    0x0648, 0x0649, 0x064a, 0x064b, 0x064c, 0x064d, 0x064e, 0x064f,
    0x0650, 0x0651, 0x0652, 0x0653, 0x0654, 0x0655, 0x0656, 0x0657,
    0x0658, 0x0659, 0x065a, 0x065b, 0x065c, 0x065d, 0x065e, 0x065f,
};

static const unsigned char iso8859_5_page04[96] = {
    0x00, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0x7a, /* 0x00-0x07 */
    0xa8, 0xa9, 0xac, 0xad, 0xae, 0xaf, /* 0x08-0x0f */
};

static int iso8859_5_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
    (void)conv; (void)n;
    unsigned char c = 0;
    if (wc < 0x00a0) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x00a0 && wc < 0xb0)
        c = iso8859_5_page00[wc-0x00a0];
    else if (wc >= 0x0400 && wc < 0x0460)
        c = iso8859_5_page04[wc-0x0400];
    else if (wc == 0x2116)
        c = 0xf0;
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}

#define NEED_TOWC /*

#define NEED_TOMB /*

#define NEED_TOWC

#define NEED_TOMB /*

#define NEED_TOWC */

#define NEED_TOMB */

13.281 iso8859_6.h


#ifndef NEED_TOWC

static const unsigned short iso8859_6_2uni[96] = {
    0x00a0, 0xfffd, 0xfffd, 0xfffd, 0x00a4, 0xfffd, 0xfffd, 0xfffd,
    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0x060c, 0x00ad, 0xfffd, 0xfffd,
    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
    0x0630, 0x0631, 0x0632, 0x0633, 0x0634, 0x0635, 0x0636, 0x0637,
    0x0638, 0x0639, 0x063a, 0x063b, 0x063c, 0x063d, 0x063e, 0x063f,
    0x0640, 0x0641, 0x0642, 0x0643, 0x0644, 0x0645, 0x0646, 0x0647,
    0x0648, 0x0649, 0x064a, 0x064b, 0x064c, 0x064d, 0x064e, 0x064f,
    0x0650, 0x0651, 0x0652, 0x0653, 0x0654, 0x0655, 0x0656, 0x0657,
    0x0658, 0x0659, 0x065a, 0x065b, 0x065c, 0x065d, 0x065e, 0x065f,
};

 Generated by Doxygen
#iso8859_7.h

```c
00028
00029 static int
00030 iso8859_6_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
00031 {
00032 unsigned char c = *s;
00033 if (c < 0xa0) {
00034 *pwc = (ucs4_t) c;
00035 return 1;
00036 }
00037 else {
00038 unsigned short wc = iso8859_6_2uni[c-0xa0];
00039 if (wc != 0xfffd) {
00040 *pwc = (ucs4_t) wc;
00041 return 1;
00042 }
00043 }
00044 return RET_ILSEQ;
00045 }
00046 #endif /* NEED_TOWC */
00047
00048 #ifdef NEED_TOMB
00049 static const unsigned char iso8859_6_page00[16] = {
00050 0xa0, 0x00, 0x00, 0x00, 0xa4, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
00051 0x00, 0x00, 0x00, 0x00, 0x00, 0xad, 0x00, 0x00, /* 0xa8-0xaf */
00052 };;
00053 static const unsigned char iso8859_6_page06[80] = {
00054 0x00, 0x00, 0x00, 0x00, 0xac, 0x00, 0x00, 0x00, /* 0x08-0x0f */
00055 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x17 */
00056 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
00057 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
00058 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x28-0x2f */
00059 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x30-0x37 */
00060 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x38-0x3f */
00061 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x40-0x4f */
00062 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x50-0x5f */
00063 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x7f */
00064 };;
00065
00066 static int
00067 iso8859_6_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
00068 {
00069 (void)conv; (void)n;
00070 unsigned char c = 0;
00071 if (wc < 0x00a0) {
00072 *r = wc;
00073 return 1;
00074 } else if (wc >= 0x00a0 && wc < 0x00b0)
00075 c = iso8859_6_page00[wc-0x00a0];
00076 if (c != 0) {
00077 *r = c;
00078 return 1;
00079 }
00080 *r = c;
00081 return 1;
00082 }
00083 return RET_ILSEQ;
00084 }
00085 #endif /* NEED_TOMB */

13.282 iso8859_7.h

```
static int iso8859_7_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
  unsigned char c = *s;
  if (c < 0xa0) {
    *pwc = (ucs4_t) c;
    return 1;
  } else {
    unsigned short wc = iso8859_7_2uni[c-0xa0];
    if (wc != 0xfffd) {
      *pwc = (ucs4_t) wc;
      return 1;
    }
  }
  return RET_ILSEQ;
}

#endif /* NEED_TOWC */

#ifdef NEED_TOMB
static const unsigned char iso8859_7_page00[32] = {
  0xa0, 0x00, 0x00, 0xa3, 0x00, 0x00, 0xa6, 0xa7, /* 0xa0-0xa7 */
  0xa8, 0xa9, 0x00, 0xab, 0xac, 0xad, 0x00, 0x00, /* 0xa8-0xaf */
  0xb0, 0xb1, 0xb2, 0xb3, 0x00, 0x00, 0xb7, /* 0xb0-0xb7 */
  0x00, 0x00, 0xb8, 0xb9, 0x00, 0xbf, 0x00, 0x00, /* 0xb8-0xbf */
};

static const unsigned char iso8859_7_page03[80] = {
  0x00, 0x00, 0x00, 0x00, 0xb4, 0xb5, 0xb6, 0xb7, /* 0x80-0x87 */
  0xb8, 0xb9, 0x00, 0xbc, 0x00, 0xbb, 0xbf, /* 0x88-0x8f */
  0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, /* 0x90-0x97 */
  0xc8, 0xc9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf, /* 0x98-0x9f */
  0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0xb0-0xb7 */
  0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* 0xb8-0xbf */
  0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* 0xc0-0xc7 */
  0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0x00, /* 0xc8-0xcf */
};

static const unsigned char iso8859_7_page20[16] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x17 */
  0x01, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x28-0x2f */
};
#endif /* NEED_TOMB */

static int iso8859_7_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
  unsigned char c = 0;
  if (wc < 0x00a0) {
    *r = wc;
    return 1;
  } else if (wc >= 0x00a0 && wc < 0x00c0) c = iso8859_7_page00[wc-0x00a0];
  else if (wc >= 0x00c0 && wc < 0x0100) c = iso8859_7_page03[wc-0x00c0];
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}
#endif /* NEED_TOMB */

13.283  iso8859_8.h
static int iso8859_8_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
  unsigned char c = *s;
  if (c >= 0xa0) {
    unsigned short wc = iso8859_8_2uni[c-0xa0];
    if (wc != 0xfffd) {
      *pwc = (ucs4_t) wc;
      return 1;
    }
  } else {
    *pwc = (ucs4_t) c;
    return 1;
  }
  return RET_ILSEQ;
}

/* NEED_TOWC */

static const unsigned char iso8859_8_page00[88] = {
  0xa0, 0x00, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, /* 0xa0-0xa7 */
  0xa8, 0xa9, 0xa0, 0xab, 0xac, 0xad, 0xae, 0xaf, /* 0xa8-0xaf */
  0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, /* 0xb0-0xb7 */
  0xb8, 0xb9, 0x00, 0xbb, 0xbc, 0xbd, 0xbe, 0x00, /* 0xb8-0xbf */
  0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, /* 0xc0-0xc7 */
  0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, /* 0xc8-0xcf */
  0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, /* 0xd0-0xd7 */
  0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf, /* 0xd8-0xdf */
  0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0xe0-0xe7 */
  0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* 0xe8-0xef */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x0f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x17 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
};

static const unsigned char iso8859_8_page05[32] = {
  0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0xe0-0xe7 */
  0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* 0xe8-0xef */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x0f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x17 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
};

static const unsigned char iso8859_8_page20[16] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x0f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x17 */
};

static int iso8859_8_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
  (void)conv; (void)n;
  unsigned char c = 0;
  if (wc < 0x00a0) {
    *r = wc;
    return 1;
  } else if (wc >= 0x00a0 && wc < 0x00f8) {
    c = iso8859_8_2uni[wc-0x00a0];
    if (c == 0xfffd) {
      *r = (ucs4_t) wc;
      return 1;
    }
  } else if (wc >= 0x05d0 && wc <= 0x05fd) {
    c = iso8859_8_2uni[wc-0x05d0];
    if (c == 0xfffd) {
      *r = (ucs4_t) wc;
      return 1;
    }
  } else if (wc >= 0x05f5 && wc < 0x0601) {
    c = iso8859_8_2uni[wc-0x05f5];
    if (c == 0xfffd) {
      *r = (ucs4_t) wc;
      return 1;
    }
  } /* t */
  return RET_ILSEQ;
}

/* NEED_TOMB */

static int iso8859_8_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
  (void)conv; (void)n;
  unsigned char c = 0;
  if (wc < 0x00a0) {
    *r = wc;
    return 1;
  } else if (wc >= 0x00a0 && wc < 0x00f8) {
    c = iso8859_8_2uni[wc-0x00a0];
    if (c == 0xfffd) {
      *r = (ucs4_t) wc;
      return 1;
    }
  } else if (wc >= 0x05d0 && wc <= 0x05fd) {
    c = iso8859_8_2uni[wc-0x05d0];
    if (c == 0xfffd) {
      *r = (ucs4_t) wc;
      return 1;
    }
  } else if (wc >= 0x05f5 && wc < 0x0601) {
    c = iso8859_8_2uni[wc-0x05f5];
    if (c == 0xfffd) {
      *r = (ucs4_t) wc;
      return 1;
    }
  } /* t */
  return RET_ILSEQ;
}

/* NEED_TOMB */
`#ifdef NEED_TOWC
static const unsigned short iso8859_9_2uni[48] = {
    /* 0xd0 */
    0x011e, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
    0x00d8, 0x00d9, 0x00da, 0x00db, 0x00dc, 0x0130, 0x015e, 0x00df,
    /* 0xe0 */
    0x00e0, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7,
    0x00e8, 0x00e9, 0x00ea, 0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef,
    /* 0xf0 */
    0x011f, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x00f7,
    0x00f8, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x0131, 0x015f, 0x00ff,
};

static int iso8859_9_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c >= 0xd0)
        *pwc = (ucs4_t) iso8859_9_2uni[c-0xd0];
    else
        *pwc = (ucs4_t) c;
    return 1;
}
#endif /* NEED_TOWC */

`#ifdef NEED_TOMB
static const unsigned char iso8859_9_page00[48] = {
    0x00, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, /* 0xd0-0xd7 */
    0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xe0, 0xe1, 0xe2, /* 0xe0-0xe2 */
    0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0xe3-0xe7 */
    0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* 0xe8-0xef */
    0xe0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* 0xf0-0xf7 */
    0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xe0, 0xe1, /* 0xf8-0xff */
};

static const unsigned char iso8859_9_page01[72] = {
    0x00, 0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f, /* 0x18-0x1f */
    0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, /* 0x20-0x27 */
    0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f, /* 0x28-0x2f */
    0x00, 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, /* 0x30-0x37 */
    0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f, /* 0x38-0x3f */
};

static int iso8859_9_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    (void)conv; (void)n;
    unsigned char c = 0;
    if (wc < 0x00d0) {
        *r = wc;
        return 1;
    } else if (wc >= 0x00d0 && wc < 0x100)
        c = iso8859_9_page00[wc-0x00d0];
    else if (wc >= 0x118 && wc < 0x160)
        c = iso8859_9_page01[wc-0x118];
    if (c != 0) {
        *r = c;
        return 1;
    } else return RET_ILSEQ;
}
#endif /* NEED_TOMB */
static int iso8859_9e_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c >= 0xa0)
        *pwc = (ucs4_t) iso8859_9e_2uni[c-0xa0];
    else
        *pwc = (ucs4_t) c;
    return 1;
}

static const unsigned char iso8859_9e_page00[96] = {
    0xa0, 0x00, 0xa2, 0xa3, 0x00, 0xa5, 0x00, 0xa7,/* 0xa0-0xa7 */
    0x00, 0xa9, 0x00, 0xab, 0x00, 0xae, 0xe5, /* 0xa8-0xaf */
    0xb0, 0x00, 0xb2, 0xb3, 0xb4, 0xb5, 0x00, 0xb7,/* 0xb0-0xb7 */
    0x00, 0xb9, 0x00, 0xbb, 0x00, 0xbd, 0xe7, /* 0xb8-0xbf */
    0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0x00, 0xc7,/* 0xc0-0xc7 */
    0xc8, 0xc9, 0xca, 0xcb, 0xc2, 0xcd, 0xce, 0xcf,/* 0xc8-0xcf */
    0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0x00, 0xdf,/* 0xd8-0xdf */
    0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe7, 0xe9, /* 0xe0-0xe7 */
    0xe8, 0xe9, 0xea, 0xeb, 0xe2, 0xe4, 0xe5, 0x00,/* 0xe8-0xef */
    0x00, 0x00, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0x00, /* 0xf0-0xf7 */
    0x00, 0xf9, 0xfa, 0xfb, 0xf2, 0xf7, 0x00, 0xfc, /* 0xf8-0xff */
};

static const unsigned char iso8859_9e_page01[136] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xd0, 0xf0,/* 0x18-0x1f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,/* 0x20-0x27 */
    0x00, 0x00, 0x00, 0x00, 0xa6, 0xb6, 0x00, 0x00,/* 0x28-0x2f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,/* 0x30-0x37 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,/* 0x38-0x3f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,/* 0x40-0x47 */
    0x00, 0x00, 0xac, 0xb6, 0x00, 0x00, 0x00, 0x00,/* 0x48-0x4f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,/* 0x50-0x57 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,/* 0x58-0x5f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,/* 0x60-0x67 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,/* 0x68-0x6f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,/* 0x70-0x77 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,/* 0x78-0x7f */
};

static const unsigned char iso8859_9e_page01_d[24] = {
    0x00, 0xaf, 0xbf, 0x00, 0x00, 0x00, 0x00, 0x00,/* 0xd0-0xd7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,/* 0xd8-0xdf */
    0x00, 0x00, 0x00, 0x00, 0xa4, 0xb4, 0x00, 0x00,/* 0xe0-0xe7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,/* 0xe8-0xef */
};

static int iso8859_9e_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    unsigned char c = 0;
    if (wc < 0x00a0) {
        *r = wc;
        return 1;
    } else if (wc < 0x00a0) {
        *r = wc;
        return 1;
    } else if (wc != 0) { Generated by Doxygen
```c
*r = c;
return 1;
}
return RET_ILSEQ;
```

### 13.286 jisx0201.h

```c
#ifdef NEED_TOWC
static int
jisx0201_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c < 0x80) {
        if (c == 0x5c)
            *pwc = (ucs4_t) 0x00a5;
        else if (c == 0x7e)
            *pwc = (ucs4_t) 0x203e;
        else
            *pwc = (ucs4_t) c;
        return 1;
    } else {
        if (c >= 0xa1 && c < 0xe0) {
            *pwc = (ucs4_t) c + 0xfec0;
            return 1;
        }
        return RET_ILSEQ;
    }
#endif /* NEED_TOWC */

#ifdef NEED_TOMB
static int
jisx0201_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    (void)conv; (void)n;
    if (wc < 0x0080 && !(wc == 0x005c || wc == 0x007e)) {
        *r = wc;
        return 1;
    } else if (wc == 0x00a5) {
        *r = 0x5c;
        return 1;
    } else if (wc == 0x203e) {
        *r = 0x7e;
        return 1;
    } else if (wc >= 0xff61 && wc < 0xffa0) {
        *r = wc - 0xfec0;
        return 1;
    }
    return RET_ILSEQ;
#endif /* NEED_TOMB */
```

### 13.287 jisx0208.h

```c
static const unsigned short jisx0208_2uni_page21[690] = {
    /* 0x21 */
```

Generated by Doxygen
static const unsigned short jisx0208_2uni_page30[6398] = {

...}

...}

Generated by Doxygen
if (i < 7808)  
} else {  
    wc = jisx0208_2uni_page21[i];  
    if (i < 690)  
        unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);  
        if (c2 >= 0x21 && c2 < 0x7f) {  
            if (n >= 2) {  
                if ((c1 >= 0x21 && c1 <= 0x28) || (c1 >= 0x30 && c1 <= 0x74)) {  
                    unsigned char c1 = (s[0] & 0x7F);  
                    if (c1 == 0x3c) {  
                        *pwc = 0x23;  
                    } else {  
                        *pwc = 0x22;  
                    }  
                } else if (i < 1410) {  
                    if (i < 8708)  
                        wc = jisx0208_2uni_page21[i];  
                    else  
                        if (wc != 0xfffd) {  
                            *pwc = (wcs4_t) wc;  
                            return 2;  
                        }  
                }  
            } else if (c2 == 0x21 && c2 < 0x7f) {  
                unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);  
                if (i < 7808)  
                    wc = jisx0208_2uni_page30[i-1410];  
                else  
                    if (wc != 0xfffd) {  
                        *pwc = (wcs4_t) wc;  
                        return 2;  
                    }  
            }  
        }  
    }  
}
13.287 jisx0208.h 2563

01932 static const Summary16 jisx0208_uni2indx_page30[16] = {
01933
01934 / * 0x3000 */
01935
01936 / * 0x4000 */
01937
01938 / * 0x5000 */
01939
01940 / * 0x6000 */
01941
01942 / * 0x7000 */
01943
01944 / * 0x8000 */
01945
01946 / * 0x9000 */
01947
01948 / * 0xA000 */
01949
01950 / * 0xB000 */
01951
01952 / * 0xC000 */
01953
01954 / * 0xD000 */
01955
01956 / * 0xE000 */
01957
01958 / * 0xF000 */
01959
01960 / * 0x1000 */
01961
01962 / * 0x1100 */
01963
01964 / * 0x1200 */
01965
01966 / * 0x1300 */
01967
01968 / * 0x1400 */
01969
01970 / * 0x1500 */
01971
01972 / * 0x1600 */
01973
01974 / * 0x1700 */
01975
01976 / * 0x1800 */
01977
01978 / * 0x1900 */
01979
01980 / * 0x1A00 */
01981
01982 / * 0x1B00 */
01983
01984 / * 0x1C00 */
01985
01986 / * 0x1D00 */
01987
01988 / * 0x1E00 */
01989
01990 / * 0x1F00 */
01991
01992 / * 0x2000 */
01993
01994 / * 0x2100 */
01995
01996 / * 0x2200 */
01997
01998 / * 0x2300 */
01999
02000 / * 0x2400 */
02001
02002 / * 0x2500 */
02003
02004 / * 0x2600 */
02005
02006 / * 0x2700 */
02007
02008 / * 0x2800 */
02009
02010 / * 0x2900 */
02011
02012 / * 0x2A00 */
02013
02014 / * 0x2B00 */
02015
02016 / * 0x2C00 */
02017
02018 / * 0x2D00 */

Generated by Doxygen
if (n >= 2) {
    ...
}
if (wc < 0x0100) 
    summary = jisx0208_uni2indx_page00[(wc>>4)];
else if (wc >= 0x0300 && wc < 0x0460) 
    summary = jisx0208_uni2indx_page03[(wc>>4)-0x030];
else if (wc >= 0x2000 && wc < 0x2320) 
    summary = jisx0208_uni2indx_page20[(wc>>4)-0x200];
else if (wc >= 0x2500 && wc < 0x2670) 
    summary = jisx0208_uni2indx_page25[(wc>>4)-0x250];
else if (wc >= 0x3000 && wc < 0x3100) 
    summary = jisx0208_uni2indx_page30[(wc>>4)-0x300];
else if (wc >= 0x4e00 && wc < 0x9fb0) 
    summary = jisx0208_uni2indx_page4e[(wc>>4)-0x4e0];
else if (wc >= 0xff00 && wc < 0xfff0) 
    summary = jisx0208_uni2indx_pageff[(wc>>4)-0xff0];
if (summary) {
    unsigned short used = summary->used;
    unsigned int i = wc & 0x0f;
    if (used & ((unsigned short) 1 « i)) {
        unsigned short c;
        /* Keep in 'used' only the bits 0..i-1. */
        used &= ((unsigned short) 1 « i) - 1;
        /* Add 'summary->indx' and the number of bits set in 'used'. */
        used = (used & 0x5555) + ((used & 0xaaaa) » 1);
        used = (used & 0x3333) + ((used & 0xcccc) » 2);
        used = (used & 0x0f0f) + ((used & 0xf0f0) » 4);
        used = (used & 0x00ff) + (used » 8);
        c = jisx0208_2charset[summary->indx + used];
        r[0] = (c » 8); r[1] = (c & 0xff);
        return 2;
    }
}
return RET_ILSEQ;
return RET_TOOSMALL;
#endif /* NEED_TOMB */
static int

// Generated by Doxygen

}
02134 
02135 static int
02136 jisx0212_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
02137 {
02138 (void)conv;
02139 if (n >= 2) {
02140 const Summary16 *summary = NULL;
02141 if (wc < 0x0460)
02142 summary = &jisx0212_uni2indx_page00[(wc»4)];
02143 else if (wc >= 0x2100 && wc < 0x2130)
02144 summary = &jisx0212_uni2indx_page21[(wc»4)-0x210];
02145 else if (wc >= 0x4e00 && wc < 0x9fb0)
02146 summary = &jisx0212_uni2indx_page4e[(wc»4)-0x4e0];
02147 if (summary) {
02148 unsigned short used = summary->used;
02149 unsigned int i = wc & 0x0f;
02150 if (used & ((unsigned short) 1 « i)) {
02151 /* Keep in 'used' only the bits 0..i-1. */
02152 used &= ((unsigned short) 1 « i) - 1;
02153 /* Add 'summary->indx' and the number of bits set in 'used'. */
02154 used = (used & 0x5555) + ((used & 0xaaaa) » 1);
02155 used = (used & 0x3333) + ((used & 0xcccc) » 2);
02156 used = (used & 0x0f0f) + ((used & 0xf0f0) » 4);
02157 used = (used & 0x00ff) + (used » 8);
02158 c = jisx0212_2charset[summary->indx + used];
02159 r[0] = (c » 8); r[1] = (c & 0xff);
02160 return 2;
02161 }
02162 return RET_ILSEQ;
02163 }
02164 }
02165 return RET_TOOSMALL;
02166 }
02167 #endif /* NEED_TOMB */
02168
static const unsigned char koi8_c_page04[240] = {
  0x00, 0xb3, 0xb1, 0xb2, 0xb4, 0xb5, 0xb6, 0xb7, /* 0x00-0x07 */
  0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0x00, 0xbe, 0xbf, /* 0x08-0x0f */
  0xe1, 0xe2, 0xf7, 0xe7, 0xe4, 0xe5, 0xf6, 0xfa, /* 0x10-0x17 */
  0xe9, 0xea, 0xeb, 0xec, 0xed, 0xe0, 0xf1, /* 0x18-0x1f */
  0xf2, 0xf3, 0xf4, 0xf5, 0xe6, 0xe8, 0xe3, 0xfe, /* 0x20-0x27 */
  0xfb, 0xfd, 0xff, 0xf9, 0xf8, 0xfc, 0xe0, 0xf1, /* 0x28-0x2f */
  0xc1, 0xc2, 0xd7, 0xc7, 0xc4, 0xc5, 0xd6, 0xda, /* 0x30-0x37 */
  0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, 0xd0, /* 0x38-0x3f */
  0xd2, 0xd3, 0x0d, 0xe3, 0xe3, 0xe4, 0xe5, 0xe6, /* 0x40-0x47 */
  0xdb, 0xdd, 0xd8, 0xdc, 0xd0, 0xd1, /* 0x48-0x4f */
  0x00, 0xa3, 0xa1, 0xa2, 0xa4, 0xa5, 0xa6, 0xa7, /* 0x50-0x57 */
  0xa8, 0xa9, 0xaa, 0xab, 0xac, 0x00, 0xae, 0xaf, /* 0x58-0x5f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x67 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x6f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
  0xbd, 0xad, 0x90, 0x80, 0x00, 0x00, 0x91, 0x81, /* 0x80-0x87 */
  0x00, 0x00, 0x92, 0x82, 0x93, 0x83, 0x00, 0x00, /* 0x88-0x8f */
  0x00, 0x00, 0x94, 0x84, 0x00, 0x00, 0x95, 0x85, /* 0x90-0x97 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
};

static const unsigned char koi8_c_page22[1] = {
  0xb0, /* 0x16-0x16 */
};

int koi8_c_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  unsigned char c = 0;
  if (wc < 0x0080) {
    *r = wc;
    return 1;
  } else if (wc >= 0x00a0 && wc < 0x00a1)
    c = koi8_c_page00[wc-0x00a0];
  else if (wc >= 0x0400 && wc < 0x04ef)
    c = koi8_c_page04[wc-0x0400];
  else if (wc >= 0x2216 && wc < 0x2217)
    c = koi8_c_page22[wc-0x2216];
  if (c != 0) {
    *r = c;
    return 1;
  } else
    return RET_ILSEQ;
}

13.290 koi8_r.h

/* $XFree86: xc/lib/X11/lcUniConv/koi8_r.h,v 1.3 2000/11/29 17:40:34 dawes Exp $ */

static const unsigned char koi8_c_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  unsigned char c = 0;
  if (wc < 0x0080) {
    *r = wc;
    return 1;
  } else if (wc >= 0x00a0 && wc < 0x00a1)
    c = koi8_c_page00[wc-0x00a0];
  else if (wc >= 0x0400 && wc < 0x04ef)
    c = koi8_c_page04[wc-0x0400];
  else if (wc >= 0x2216 && wc < 0x2217)
    c = koi8_c_page22[wc-0x2216];
  if (c != 0) {
    *r = c;
    return 1;
  } else
    return RET_ILSEQ;
}

*/

13.290 koi8_r.h

/* $XFree86: xc/lib/X11/lcUniConv/koi8_r.h,v 1.3 2000/11/29 17:40:34 dawes Exp $ */

/* Specification: RFC 1489 */
static int koi8_r_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    (void)conv; (void)n;
    unsigned char c = 0;
    if (wc < 0x0080) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x00a0 && wc < 0x00f8)
        c = koi8_r_page00[wc-0x00a0];
    else if (wc >= 0x0400 && wc < 0x0458)
        c = koi8_r_page04[wc-0x0400];
    else if (wc >= 0x2218 && wc < 0x2268)
        c = koi8_r_page22[wc-0x2218];
    else if (wc >= 0x2320 && wc < 0x2328)
        c = koi8_r_page23[wc-0x2320];
    else if (wc >= 0x2500 && wc < 0x25a8)
        c = koi8_r_page25[wc-0x2500];
    else if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}
#endif /* NEED_TOMB */

13.291 koi8_u.h

/* Specification: RFC 2319 */

static const unsigned short koi8_u_2uni[128] = {
    0x2500, 0x2502, 0x2510, 0x2514, 0x2518, 0x2524,
    0x2580, 0x2584, 0x2588, 0x2590, 0x2591, 0x2592,
    0x2593, 0x2320, 0x25a0, 0x2219, 0x2248, 0x2264,
    0x2265, 0x00a0, 0x00b0, 0x00b2, 0x00f7, 0x2550,
    0x2551, 0x2552, 0x0451, 0x0454, 0x0456, 0x0457,
    0x2558, 0x2559, 0x255a, 0x255b, 0x0491, 0x255d,
    0x0401, 0x0404, 0x0406, 0x0407, 0x040c, 0x040d,
    0x040e, 0x0416, 0x0417, 0x0418, 0x0419, 0x041a,
    0x041b, 0x041c, 0x041d, 0x041e, 0x041f, 0x0420,
    0x0421, 0x0422, 0x0423, 0x0424, 0x0425, 0x0426,
    0x0427, 0x0428, 0x0429, 0x042a, 0x042b, 0x042c,
    0x042d, 0x042e, 0x042f, 0x0430, 0x0431, 0x0432,
    0x0433, 0x0434, 0x0435, 0x0436, 0x0437, 0x0438,
    0x0439, 0x043a, 0x043b, 0x043c, 0x043d, 0x043e,
    0x043f, 0x0440, 0x0441, 0x0442, 0x0443, 0x0444,
    0x0445, 0x0446, 0x0447, 0x0448, 0x0449, 0x044a,
    0x044b, 0x044c, 0x044d, 0x044e, 0x044f, 0x0450,
    0x0451, 0x0452, 0x0454, 0x0456, 0x0457, 0x0458,
    0x0459, 0x045a, 0x045b, 0x045c, 0x045d, 0x045e,
    0x045f, 0x0460, 0x0461, 0x0462, 0x0463, 0x0464,
    0x0465, 0x0466, 0x0467, 0x0468, 0x0469, 0x046a,
    0x046b, 0x046c, 0x046d, 0x046e, 0x046f, 0x0470,
    0x0471, 0x0472, 0x0473, 0x0474, 0x0475, 0x0476,
    0x0477, 0x0478, 0x0479, 0x047a, 0x047b, 0x047c,
    0x047d, 0x047e, 0x047f, 0x0480, 0x0481, 0x0482,
    0x0483, 0x0484, 0x0485, 0x0486, 0x0487, 0x0488,
    0x0489, 0x048a, 0x048b, 0x048c, 0x048d, 0x048e,
    0x048f, 0x0490, 0x0491, 0x0492, 0x0493, 0x0494,
koi8_u.h

koi8_u_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)

static unsigned char koi8_u_page25[168] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa8-0xaf */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xb0-0xb7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xc7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
}

else if (wc >= 0x0400 && wc < 0x0498)
    c = koi8_u_page00[wc-0x00a0];

else if (wc >= 0x00a0 && wc < 0x00f8)
    c = koi8_u_page22[wc-0x2218];

else if (wc >= 0x0498 && wc < 0x0598)
    c = koi8_u_page23[wc-0x2320];

else if (wc >= 0x0598 && wc < 0x05e8)
    c = koi8_u_page24[wc-0x2400];

else if (wc >= 0x05e8 && wc < 0x06a8)
    c = koi8_u_page25[wc-0x2500];

if (wc < 0x0080) {
    unsigned char c = 0;
    koi8_u_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
static const unsigned short ksc5601_2uni_page21[1115] = {
    0x2025, 0x2026, 0x00a8, 0x3003,
    #ifdef NEED_TOWC
    0x2015, 0x201F, 0xfffc, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
    #endif /* NEED_TOMB */
    return 1;
    *r = c;
    return RET_ILSEQ;
};

13.292  ksc5601.h
/ * 0x3d */

/ * 0x3e */

/ * 0x3f */

/ * 0x40 */

/ * 0x41 */

/ * 0x42 */

/ * 0x43 */

/ * 0x44 */

/ * 0x45 */

/ * 0x46 */

/ * 0x47 */

/ * 0x48 */

/ * 0x49 */

/ * 0x4a */

/ * 0x4b */

/ * 0x4c */

/ * 0x4d */

/ * 0x4e */

/ * 0x4f */

/ * 0x50 */

/ * 0x51 */

/ * 0x52 */

/ * 0x53 */

/ * 0x54 */

/ * 0x55 */

/ * 0x56 */

/ * 0x57 */

/ * 0x58 */

/ * 0x59 */

/ * 0x5a */

/ * 0x5b */

/ * 0x5c */

/ * 0x5d */

/ * 0x5e */

/ * 0x5f */

/ * 0x60 */

/ * 0x61 */

/ * 0x62 */

/ * 0x63 */

/ * 0x64 */

/ * 0x65 */

/ * 0x66 */

/ * 0x67 */

/ * 0x68 */

/ * 0x69 */

/ * 0x6a */

/ * 0x6b */

/ * 0x6c */

/ * 0x6d */

/ * 0x6e */

/ * 0x6f */

/ * 0x70 */

/ * 0x71 */

/ * 0x72 */

/ * 0x73 */

/ * 0x74 */

/ * 0x75 */

/ * 0x76 */

/ * 0x77 */

/ * 0x78 */

/ * 0x79 */

/ * 0x7a */

/ * 0x7b */

/ * 0x7c */

/ * 0x7d */

/ * 0x7e */

/ * 0x7f */
static int
ksc5601_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c1 = (s[0] & 0x7F);
    if ((c1 >= 0x21 && c1 <= 0x2c) || (c1 >= 0x30 && c1 <= 0x48) || (c1 >= 0x4a && c1 <= 0x7d)) {
        if (n >= 2) {
            unsigned char c2 = (s[1] & 0x7F);
            if (c2 >= 0x21 && c2 < 0x7f) {
                unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);
                unsigned short wc = 0xfffd;
                if (i < 1410) {
                    if (i < 1115)
                        wc = ksc5601_2uni_page21[i];
                    else if (i < 3854)
                        wc = ksc5601_2uni_page30[i-1410];
                    else {
                        if (i < 8742)
                            wc = ksc5601_2uni_page4a[i-3854];
                    }
                }
                if (wc != 0xfffd) {
                    *pwc = (ucs4_t) wc;
                    return 2;
                }
            }
        }
        return RET_ILSEQ;
    }
    return RET_TOOFEW(0);
}

static unsigned char c1 = (s[0] & 0x7F);
if ((c1 >= 0x21 && c1 <= 0x2c) || (c1 >= 0x30 && c1 <= 0x48) || (c1 >= 0x4a && c1 <= 0x7d)) {
    if (n >= 2) {
        unsigned char c2 = (s[1] & 0x7F);
        if (c2 >= 0x21 && c2 < 0x7f) {
            unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);
            unsigned short wc = 0xfffd;
            if (i < 1410) {
                if (i < 1115)
                    wc = ksc5601_2uni_page21[i];
                else if (i < 3854)
                    wc = ksc5601_2uni_page30[i-1410];
                else {
                    if (i < 8742)
                        wc = ksc5601_2uni_page4a[i-3854];
                }
            }
            if (wc != 0xfffd) {
                *pwc = (ucs4_t) wc;
                return 2;
            }
        }
    }
    return RET_ILSEQ;
}
static const Summary16 ksc5601_uni2indx_pagef9[17] = {
    { 7856, 0xffff }, { 7872, 0xffff }, { 7888, 0xffff }, { 7904, 0xffff },
    { 7920, 0xffff }, { 7936, 0xffff }, { 7952, 0xffff }, { 7968, 0xffff },
    { 7984, 0xffff }, { 8000, 0xffff }, { 8016, 0xffff }, { 8032, 0xffff },
    { 8048, 0xffff }, { 8064, 0xffff }, { 8080, 0xffff }, { 8096, 0xffff },
    { 8112, 0xffff },
};

static const Summary16 ksc5601_uni2indx_pageff[15] = {
    { 8124, 0x0fff }, { 8139, 0xffff }, { 8155, 0xffff }, { 8171, 0xffff },
    { 8187, 0xffff }, { 8203, 0x7fff }, { 8218, 0x0000 }, { 8218, 0x0000 },
    { 8218, 0x0000 }, { 8218, 0x0000 }, { 8218, 0x0000 }, { 8218, 0x0000 }, { 8218, 0x006f },
};

static int ksc5601_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    (void)conv;
    if (n >= 2) {
        const Summary16 *summary = NULL;
        if (wc < 0x0460)
            summary = &ksc5601_uni2indx_page00[(wc»4)];
        else if (wc >= 0x2000 && wc < 0x2670)
            summary = &ksc5601_uni2indx_page20[(wc»4)-0x200];
        else if (wc >= 0x3000 && wc < 0x33e0)
            summary = &ksc5601_uni2indx_page30[(wc»4)-0x300];
        else if (wc >= 0x4e00 && wc < 0x9fa0)
            summary = &ksc5601_uni2indx_page4e[(wc»4)-0x4e0];
        else if (wc >= 0xac00 && wc < 0xd7a0)
            summary = &ksc5601_uni2indx_pageac[(wc»4)-0xac0];
        else if (wc >= 0xf900 && wc < 0xfa10)
            summary = &ksc5601_uni2indx_pagef9[(wc»4)-0xf90];
        else if (wc >= 0xff00 && wc < 0xfff0)
            summary = &ksc5601_uni2indx_pageff[(wc»4)-0xff0];
        if (summary) {
            unsigned short used = summary->used;
            unsigned int i = wc & 0x0f;
            if (used & ((unsigned short) 1 « i)) {
                unsigned short c;
                /* Keep in 'used' only the bits 0..i-1. */
                used &= ((unsigned short) 1 « i) - 1;
                /* Add 'summary->indx' and the number of bits set in 'used'. */
                used = (used & 0x5555) + ((used & 0xaaaa) » 1);
                used = (used & 0x3333) + ((used & 0xcccc) » 2);
                used = (used & 0x0f0f) + (used » 8);
                c = ksc5601_2charset[summary->indx + used];
                r[0] = (c » 8); r[1] = (c & 0xff);
                return 2;
            }
        }
    } else
        return RET_ILSEQ;
    return RET_TOOSMALL;
} #endif /* NEED_TOMB */

static const unsigned short mulelao_2uni[96] = {
    { 0xa0 }, { 0xe0a0 }, { 0xe081 }, { 0xe082 }, { 0xfffd }, { 0xe084 }, { 0xfffd }, { 0xe086 },
    { 0xe088 }, { 0xfffd }, { 0xe08a }, { 0xfffd }, { 0xe08c }, { 0xfffd }, { 0xe08e },
    { 0xe090 }, { 0xfffd }, { 0xe092 }, { 0xfffd }, { 0xe094 }, { 0xe095 }, { 0xe096 }, { 0xe097 },
    { 0xfffd }, { 0xe099 }, { 0xe09a }, { 0xe09b }, { 0xe09c }, { 0xe09d }, { 0xfffd }, { 0xfffd },
    { 0xe0a1 }, { 0xe0a2 }, { 0xe0a3 }, { 0xfffd }, { 0xe0a5 }, { 0xfffd }, { 0xe0a7 },
    { 0xfffd }, { 0xe0a9 }, { 0xe0b0 }, { 0xe0b1 }, { 0xe0b2 }, { 0xe0b3 }, { 0xe0b4 }, { 0xe0b5 },
    { 0xe0b6 }, { 0xe0b7 }, { 0xe0b8 }, { 0xe0b9 }, { 0xe0b9 }, { 0xe0ba }, { 0xe0bb }, { 0xe0bc },
    { 0xe0bd }, { 0xfffd }, { 0xfffd }, { 0xe0c0 }, { 0xe0c1 }, { 0xe0c2 }, { 0xe0c3 }, { 0xe0c4 },
    { 0xfffd }, { 0xe0c5 }, { 0xfffd }, { 0xe0c8 }, { 0xe0c9 }, { 0xe0ca }, { 0xe0cb }, { 0xe0cc },
    { 0xe0cd }, { 0xfffd }, { 0xfffd },
}

13.293 mulelao.h

Generated by Doxygen
static int mulelao_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c < 0xa0) {
        *pwc = (ucs4_t) c;
        return 1;
    }
    else {
        unsigned short wc = mulelao_2uni[c-0xa0];
        if (wc != 0xfffd) {
            *pwc = (ucs4_t) wc;
            return 1;
        }
    }
    return RET_ILSEQ;
}

static const unsigned char mulelao_page0e[96] = {
    0x00, 0xa1, 0xa2, 0x00, 0xa4, 0x00, 0x00, 0xa7, / * 0x80-0x87 */
    0xa8, 0x00, 0xaa, 0x00, 0x00, 0xad, 0x00, 0x00, / * 0x88-0x8f */
    0x00, 0x00, 0x00, 0x00, 0xb4, 0xb5, 0xb6, 0xb7, / * 0x90-0x97 */
    0xb8, 0x00, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, / * 0x98-0x9f */
    0x00, 0x00, 0x00, 0x00, 0xc1, 0xc2, 0xc3, 0x00, / * 0xa0-0xa7 */
    0xc5, 0x00, 0xc7, 0x00, 0x00, 0xca, 0xcb, 0x00, / * 0xa8-0xaf */
    0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, / * 0xb0-0xb7 */
    0xd8, 0xd9, 0x00, 0xdb, 0xdc, 0xdd, 0x00, 0x00, / * 0xb8-0xbf */
    0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, / * 0xc0-0xc7 */
    0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xe0, 0xe0, / * 0xc8-0xcf */
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, / * 0xd0-0xd7 */
    0xf8, 0xf9, 0x00, 0x00, 0xfc, 0xfd, 0x00, 0x00, / * 0xd8-0xdf */
};

static int mulelao_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    unsigned char c = 0;
    if (wc < 0x00a0) {
        *r = wc;
        return 1;
    }
    else if (wc == 0x00a0)
        c = 0xa0;
    else if (wc >= 0x0e80 && wc < 0x0ee0)
        c = mulelao_page0e[wc-0x0e80];
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}

static const unsigned short tatar_cyr_2uni[128] = {
    / * 0x80 */
    0x04d8, 0x0403, 0x201a, 0x0453, 0x201e, 0x2026, 0x2020, 0x2021,
    / * 0x90 */
    0x04d9, 0x2018, 0x2019, 0x201c, 0x201d, 0x2022, 0x2013, 0x2014,
    / * 0xa0 */
    0x00a0, 0x040e, 0x044e, 0x0447, 0x040b, 0x0408, 0x0409, 0x040a,
    / * 0xb0 */
    0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7,
    / * 0xc0 */
    0x0451, 0x2116, 0x0454, 0x045b, 0x0458, 0x0455, 0x0457,
    / * 0x80 */
    0x00c0, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7,
    / * 0xe0 */
};

Generated by Doxygen
tatar_cyr_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
static int

if (c < 0x80)
	*pwc = (ucs4_t) tatar_cyr_2uni[c-0x80];
else
	*pwc = (ucs4_t) c;
if (c < 0x80)

c = tatar_cyr_page20[wc-0x2010];
else if (wc >= 0x00a0 && wc < 0x00bc)
return 1;

unsigned char c = 0;
if (wc < 0x0080)
	c = tatar_cyr_page00[32] = {
0x86, 0x87, 0x95, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa0-0x0f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x67 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x5f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x57 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x48-0x4f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x38-0x3f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x28-0x2f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x17 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x08-0x0f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x07 */
};

static const unsigned char tatar_cyr_page20[48] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x07 */
0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, /* 0x08-0x0f */
0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x10, /* 0x10-0x17 */
0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, /* 0x18-0x1f */
0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f, 0x20, /* 0x20-0x27 */
0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, /* 0x28-0x2f */
0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f, 0x30, /* 0x30-0x37 */
0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, /* 0x38-0x3f */
0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f, 0x40, /* 0x40-0x4f */
0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, /* 0x50-0x57 */
0x49, 0x4a, 0x4b, 0x4c, 0x4d, 0x4e, 0x4f, 0x50, /* 0x58-0x5f */
0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, /* 0x60-0x67 */
0x59, 0x5a, 0x5b, 0x5c, 0x5d, 0x5e, 0x5f, 0x60, /* 0x68-0x6f */
0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, /* 0x70-0x77 */
0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f, 0x70, /* 0x78-0x7f */
0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, /* 0x80-0x87 */
0x79, 0x7a, 0x7b, 0x7c, 0x7d, 0x7e, 0x7f, 0x80, /* 0x88-0x8f */
0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, /* 0x90-0x97 */
0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f, /* 0xa0-0xb7 */
0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf, /* 0xb8-0xbf */
0xbf, 0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, /* 0xc8-0xcf */
0xc7, /* 0xd8-0x16 */
};

Generated by Doxygen
00114 else if (wc /* 0x20ac */)
00115 c = 0x88;
00116 else if (wc == 0x20ac)
00117 c = tatar_cyr_page21[wc-0x2110];
00118 if (c != 0) {
00119 *r = c;
00120 return 1;
00121 }
00122 return RET_ILSEQ;
00123 }

13.295 tcvn.h

00002
00003 /* TCVN-5712 */
00004
00005 static const unsigned short tcvn_2uni_1[32] = {
00006 / * 0x00 */
00007 0x0000, 0x00da, 0x1ee4, 0x0003, 0x1eea, 0x1eec, 0x1eee, 0x0007,
00008 0x0008, 0x0009, 0x000a, 0x000b, 0x000c, 0x000d, 0x000e, 0x000f,
00009 / * 0x10 */
00010 0x0010, 0x1ee8, 0x1ef0, 0x1ef2, 0x1ef6, 0x1ef8, 0x00dd, 0x1ef4,
00011 0x0018, 0x0019, 0x001a, 0x001b, 0x001c, 0x001d, 0x001e, 0x001f,
00012 / * 0x80 */
00013 0x00c0, 0x1ea2, 0x00c3, 0x00c1, 0x1ea0, 0x1eb6, 0x1ebc, 0x00c8,
00014 0x00c9, 0x00ca, 0x00cb, 0x00cc, 0x00cd, 0x00ce, 0x00cf, 0x00d0,
00015 / * 0x90 */
00016 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7, 0x00d8, 0x00d9,
00017 0x00da, 0x00db, 0x00dc, 0x00dd, 0x00de, 0x00df, 0x00e0, 0x00e1,
00018 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7, 0x00e8, 0x00e9,
00019 / * 0xb0 */
00020 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7, 0x00b8, 0x00b9,
00021 0x00ba, 0x00bb, 0x00bc, 0x00bd, 0x00be, 0x00bf, 0x00c0, 0x00c1,
00022 / * 0xc0 */
00023 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7, 0x00c8, 0x00c9,
00024 0x00ca, 0x00cb, 0x00cc, 0x00cd, 0x00ce, 0x00cf, 0x00d0, 0x00d1,
00025 / * 0xda */
00026 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7, 0x00d8, 0x00d9,
00027 0x00da, 0x00db, 0x00dc, 0x00dd, 0x00de, 0x00df, 0x00e0, 0x00e1,
00028 / * 0xe0 */
00029 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7, 0x00e8, 0x00e9,
00030 0x00ea, 0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef, 0x00f0, 0x00f1,
00031 / * 0xf0 */
00032 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x00f7, 0x00f8, 0x00f9,
00033 / * 0x0100 */
00034 0x0101, 0x0102, 0x0103, 0x0104, 0x0105, 0x0106, 0x0107, 0x0108,
00035 / * 0x0110 */
00036 0x0111, 0x0112, 0x0113, 0x0114, 0x0115, 0x0116, 0x0117, 0x0118,
00037 / * 0x0120 */
00038 0x0121, 0x0122, 0x0123, 0x0124, 0x0125, 0x0126, 0x0127, 0x0128,
00039 / * 0x0130 */
00040 0x0131, 0x0132, 0x0133, 0x0134, 0x0135, 0x0136, 0x0137, 0x0138,
00041 / * 0x0140 */
00042 static int
tcvn_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
00043 { return 1; }

Generated by Doxygen
/* $XFree86: xc/lib/X11/lcUniConv/tis620.h,v 1.4 2001/02/09 00:02:54 dawes Exp $ */

13.296 tis620.h

00001 /* $XFree86: xc/lib/X11/lcUniConv/tis620.h,v 1.4 2001/02/09 00:02:54 dawes Exp $ */
00002
00003 /*
00004 */
00005 /*
00006 */
00007 /* TIS620-0 */
00008 /*
00009 */
00010
00011 static int
00012 tcvn_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
00013 {
00014    unsigned char c = 0;
00015
00016    if (wc < 0x0080 && (wc >= 0x0020 || (0x00fe0076 & (1 « wc)) == 0)) {
00017        *r = wc;
00018        return 1;
00019    }
00020
00021    else if (wc >= 0x0300 && wc < 0x0328)
00022        c = tcvn_page03[wc-0x0300];
00023
00024    else if (wc >= 0x1ea0 && wc < 0x1f00)
00025        c = tcvn_page1e[wc-0x1ea0];
00026
00027    if (c != 0) {
00028        *r = c;
00029        return 1;
00030    }
00031    return RET_ILSEQ;
00032
00033 }
00034
00035 static int
00036 tcvn_2uniw (conv_t conv, ucs4_t wc, unsigned char *r, int n)
00037 {
00038    unsigned char c = 0;
00039
00040    if (wc < 0x0080 && (wc >= 0x0020 || (0x00fe0076 & (1 « wc)) == 0)) {
00041        *r = wc;
00042        return 1;
00043    }
00044
00045    else if (wc >= 0x0300 && wc < 0x0328)
00046        c = tcvn_page03[wc-0x0300];
00047
00048    else if (wc >= 0x1ea0 && wc < 0x1f00)
00049        c = tcvn_page1e[wc-0x1ea0];
00050
00051    if (c != 0) {
00052        *r = c;
00053        return 1;
00054    }
00055    return RET_ILSEQ;
00056
00057 

Generated by Doxygen
static int tis620_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c < 0x80) {
        *pwc = (ucs4_t) c;
        return 1;
    } else if (c < 0xa0) {
    } else {
        unsigned short wc = tis620_2uni[c-0xa0];
        if (wc != 0xfffd) {
            *pwc = (ucs4_t) wc;
            return 1;
        }
    }
    return RET_ILSEQ;
}

static const unsigned char tis620_page0e[96] = {
    0x00, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, /* 0x00-0x07 */
    0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf, /* 0x08-0x0f */
    0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, /* 0x10-0x17 */
    0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf, /* 0x18-0x1f */
    0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, /* 0x20-0x27 */
    0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xd0, 0xca, 0xcf, /* 0x28-0x2f */
    0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, /* 0x30-0x37 */
    0xd8, 0xd9, 0xda, 0xcc, 0xe0, 0xe0, 0xe0, 0xef, /* 0x38-0x3f */
    0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0x40-0x47 */
    0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xe8, 0xf0, /* 0x48-0x4f */
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* 0x50-0x57 */
    0xf8, 0xf9, 0xfa, 0xfb, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x5f */
};

static int tis620_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    unsigned char c = 0;
    if (wc < 0x0080) {
        *r = wc;
        return 1;
    } else if (wc >= 0x0e00 && wc < 0x0e60) c = tis620_2uni[wc-0x0e00];
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}

static const unsigned char tis620_page0e[96] = {
    0x00, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, /* 0x00-0x07 */
    0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf, /* 0x08-0x0f */
    0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, /* 0x10-0x17 */
    0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf, /* 0x18-0x1f */
    0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, /* 0x20-0x27 */
    0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xd0, 0xca, 0xcf, /* 0x28-0x2f */
    0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, /* 0x30-0x37 */
    0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xe8, 0xf0, /* 0x48-0x4f */
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* 0x50-0x57 */
    0xf8, 0xf9, 0xfa, 0xfb, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x5f */
};

static int ucs2be_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    if (n >= 2) {
        if (s[0] >= 0xd8 && s[0] < 0xe0) {
            return RET_ILSEQ;
        } else {
            *pwc = (s[0] « 8) + s[1];
            return 2;
        }
    }
    return RET_TOOFEW(0);
}

static int ucs2be_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    if (wc < 0x10000 && wc < 0xe000) {
        return RET_TOOFEW(0);
    }
    return RET_ILSEQ;
}
utf8.h

utf8_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)

if (c < 0x80) {
  *pwc = c;
  return 1;
} else if (c < 0xc2) {
  return RET_ILSEQ;
} else if (c < 0xe0) {
  if (n < 2)
    return RET_TOOFEW(0);
  if (!((s[1] ^ 0x80) < 0x40))
    return RET_ILSEQ;
  *pwc = ((ucs4_t) (c & 0x1f) « 6)
    | (ucs4_t) (s[1] ^ 0x80);
  return 2;
} else if (c < 0xf0) {
  if (n < 3)
    return RET_TOOFEW(0);
  if (!((s[1] ^ 0x80) < 0x40 && (s[2] ^ 0x80) < 0x40)
    && (c >= 0xe1 || s[1] >= 0xa0))
    return RET_ILSEQ;
  *pwc = ((ucs4_t) (c & 0x0f) « 12)
    | ((ucs4_t) (s[1] ^ 0x80) « 6)
    | (ucs4_t) (s[2] ^ 0x80);
  return 3;
} else if (c < 0xf8) {
  if (n < 4)
    return RET_TOOFEW(0);
  if (!((s[1] ^ 0x80) < 0x40 && (s[2] ^ 0x80) < 0x40
    && (s[3] ^ 0x80) < 0x40)
    && (c >= 0xf1 || s[1] >= 0x90))
    return RET_ILSEQ;
  *pwc = ((ucs4_t) (c & 0x07) « 18)
    | ((ucs4_t) (s[1] ^ 0x80) « 12)
    | ((ucs4_t) (s[2] ^ 0x80) « 6)
    | (ucs4_t) (s[3] ^ 0x80);
  return 4;
} else if (c < 0xff) {
  if (n < 5)
    return RET_TOOFEW(0);
  if (!((s[1] ^ 0x80) < 0x40 && (s[2] ^ 0x80) < 0x40
    && (s[3] ^ 0x80) < 0x40 && (s[4] ^ 0x80) < 0x40)
    && (c >= 0xfd || s[1] >= 0x88))
    return RET_ILSEQ;
  *pwc = ((ucs4_t) (c & 0x03) « 24)
    | ((ucs4_t) (s[1] ^ 0x80) « 18)
    | ((ucs4_t) (s[2] ^ 0x80) « 12)
    | ((ucs4_t) (s[3] ^ 0x80) « 6)
    | (ucs4_t) (s[4] ^ 0x80);
  return 5;
} else if (c < 0xfe) {
  if (n < 6)
    return RET_TOOFEW(0);
  if (!((s[1] ^ 0x80) < 0x40 && (s[2] ^ 0x80) < 0x40
    && (s[3] ^ 0x80) < 0x40 && (s[4] ^ 0x80) < 0x40
    && (s[5] ^ 0x80) < 0x40)
    && (c >= 0xfd || s[1] >= 0x84))
    return RET_ILSEQ;
  *pwc = ((ucs4_t) (c & 0x01) « 30)
    | ((ucs4_t) (s[1] ^ 0x80) « 24)
    | ((ucs4_t) (s[2] ^ 0x80) « 18)
    | ((ucs4_t) (s[3] ^ 0x80) « 12)
    | ((ucs4_t) (s[4] ^ 0x80) « 6)
  return 6;
00002
00003 /
00004 * VISCII1.1-1
00005 */
00006
00007 /* Specification: RFC 1456 */
00008
00009 static const unsigned short viscii_2uni_1[32] = {
00010 /* 0x00 */
00011 0x0000, 0x0001, 0x1eb2, 0x0003, 0x0004, 0x1eb4, 0x1eaa, 0x0007, 00012 0x0008, 0x0009, 0x000a, 0x000b, 0x000c, 0x000d, 0x000e, 0x000f,
00013 /* 0x10 */
00014 0x0010, 0x0011, 0x0012, 0x0013, 0x1ef6, 0x0015, 0x0016, 0x0017, 00015 0x0018, 0x1ef8, 0x001a, 0x001b, 0x001c, 0x001d, 0x001e, 0x001f,
00016 };
00017 static const unsigned short viscii_2uni_2[128] = {
00018 /* 0x80 */
00019 0x1ea0, 0x1eae, 0x1eb0, 0x1eb6, 0x1ea4, 0x1ea6, 0x00e2, 0x1eb8, 00020 0x01eb, 0x01eb, 0x01ec, 0x01ed, 0x01ec, 0x01ec, 0x01ed, 0x01ed,
00021 /* 0x90 */
00022 0x01ed, 0x01ed, 0x01ed, 0x01ed, 0x01ed, 0x01ed, 0x01ed, 0x01ed,
00023 0x01eb, 0x01eb, 0x01eb, 0x01eb, 0x01eb, 0x01eb, 0x01eb, 0x01eb,
00024 /* 0xa0 */
00025 0x01eb, 0x01eb, 0x01eb, 0x01eb, 0x01eb, 0x01eb, 0x01eb, 0x01eb,
00026 0x01eb, 0x01eb, 0x01eb, 0x01eb, 0x01eb, 0x01eb, 0x01ed, 0x01ed,
00027 /* 0xb0 */
00028 0x01ed, 0x01ed, 0x01ed, 0x01ed, 0x01ed, 0x01ed, 0x01ed, 0x01ed,
00029 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec,
00030 /* 0xc0 */
00031 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec,
00032 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec,
00033 /* 0xd0 */
00034 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec,
00035 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec,
00036 /* 0xe0 */
00037 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec,
00038 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec,
00039 /* 0xf0 */
00040 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec,
00041 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec, 0x01ec,
00042 };
# mk_wcwidth.c

```c
/*
 * FLTK: Important!
 * This file should remain as close to Markus Kuhn's original source
 * as possible for easy checking for changes later, however unlikely.
 * All customisations to work with FLTK shall be annotated!
*/

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <wchar.h>

#define RET_ILSEQ 1

static const unsigned char viscii_page00[64+184] = {
  0xc0, 0xc1, 0xc2, 0xc3, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xc7 */
  0xc8, 0xc9, 0xca, 0x00, 0xcc, 0xcd, 0x00, 0x00, /* 0xc8-0xcf */
  0x00, 0x00, 0xd2, 0xd3, 0xd4, 0xa0, 0x00, 0x00, /* 0xd0-0xd7 */
  0x00, 0x00, 0xd9, 0xda, 0x00, 0x00, 0xdd, 0x00, /* 0xd8-0xdf */
  0xe0, 0xe1, 0xe2, 0xe3, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xe7 */
  0xe8, 0xe9, 0xea, 0x00, 0xec, 0xed, 0x00, 0x00, /* 0xe8-0xef */
  0x00, 0x00, 0xf2, 0xf3, 0xf4, 0xf5, 0x00, 0x00, /* 0xf0-0xf7 */
  0x00, 0x00, 0xf9, 0xfa, 0x00, 0x00, 0xfd, 0x00, /* 0xf8-0xff */
  0x00, 0x00, 0xc5, 0xe5, /* 0x00-0x07 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x08-0x0f */
  0xd0, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x17 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x28-0x2f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x30-0x37 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x38-0x3f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x40-0x47 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x48-0x4f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x50-0x57 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x5f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x67 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x6f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
  0xb4, 0xbd, /* 0xa0-0xa7 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa8-0xaf */
  0xdf, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xb0-0xb7 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xf8-0xff */
};

static const unsigned char viscii_page1e[96] = {
  0x80, 0xd5, 0xc4, 0xe4, 0x84, 0xa4, 0x85, 0xa5, /* 0xa0-0xa7 */
  0x86, 0xa6, 0x06, 0xe7, 0x87, 0xa7, 0x81, 0xa1, /* 0xa8-0xaf */
  0x82, 0xa2, 0x02, 0xc6, 0x05, 0xc7, 0x83, 0xa3, /* 0xb0-0xb7 */
  0x89, 0xa9, 0xcb, 0xeb, 0x88, 0xa8, 0x8a, 0xaa, /* 0xb8-0xbf */
  0x8b, 0xab, 0x8c, 0xac, 0x8d, 0xad, 0x8e, 0xae, /* 0xc0-0xc7 */
  0x9b, 0xef, 0x98, 0xb8, 0x9a, 0xf7, 0x99, 0xf6, /* 0xc8-0xcf */
  0x8f, 0xaf, 0x90, 0xb0, 0x91, 0xb1, 0x92, 0xb2, /* 0xd0-0xd7 */
  0x93, 0xb5, 0x95, 0xbe, 0x96, 0xb6, 0x97, 0xb7, /* 0xd8-0xdf */
  0xb3, 0xd1, 0xbb, 0xd7, 0xbc, 0xd8, 0xff, 0xe6, /* 0xe0-0xe7 */
  0xb9, 0xf1, 0x9f, 0xcf, 0x16, 0xdc, 0x14, 0xd6, /* 0xf0-0xff */
  0x19, 0xdb, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe8-0xff */
};

static int
viscii_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  unsigned char c = 0;
  if (wc < 0x0080 && (wc >= 0x0020 || (0x42100064 & (1 << wc)) == 0)) {
    *r = wc;
    return 1;
  } else if (wc >= 0x00c0 && wc < 0x1b8)
    c = viscii_page00[wc-0x00c0];
  else if (wc >= 0x1ea0 && wc < 0x1f00)
    c = viscii_page1e[wc-0x1ea0];
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}

13.300 mk_wcwidth.c

```
This is an implementation of wcwidth() and wcswidth() (defined in IEEE Std 1002.1-2001) for Unicode.

In fixed-width output devices, Latin characters all occupy a single "cell" position of equal width, whereas ideographic CJK characters occupy two such cells. Interoperability between terminal-line applications and (teletype-style) character terminals using the UTF-8 encoding requires agreement on which character should advance how many cell positions. No established formal standards exist at present on which Unicode character shall occupy how many cell positions on character terminals. These routines are a first attempt of defining such behavior based on simple rules applied to data provided by the Unicode Consortium.

For some graphical characters, the Unicode standard explicitly defines a character-cell width via the definition of the East Asian FullWidth (F), Wide (W), Half-width (H), and Narrow (Na) classes. Class, the width choice depends purely on a preference of backward compatibility with either historic CJK or Western practice. Choosing single-width for these characters is easy to justify as the appropriate long-term solution, as the CJK practice of displaying these characters as double-width comes from historic implementation simplicity (8-bit encoded characters were displayed single-width and 16-bit ones double-width, even for Greek, Cyrillic, etc.) and not any typographic considerations.

Much less clear is the choice of width for the Not East Asian (Neutral) class. Existing practice does not dictate a width for any of these characters. It would nevertheless make sense typographically to allocate two character cells to characters such as for instance EM SPACE or VOLUME INTEGRAL, which cannot be represented adequately with a single-width glyph. The following routines at present merely assign a single-cell width to all neutral characters, in the interest of simplicity. This is not entirely satisfactory and should be reconsidered before establishing a formal standard in this area. At the moment, the decision which Not East Asian (Neutral) characters should be represented by double-width glyphs cannot yet be answered by applying a simple rule from the Unicode database content. Setting up a proper standard for the behavior of UTF-8 character terminals will require a careful analysis not only of each Unicode character, but also of each presentation form, something the author of these routines has avoided to do so far.

FLTK - avoid possible problems on systems with 32-bit wchar_t. Don't include wchar.h, and change wchar_t to unsigned int.

Generated by Doxygen
while (max >= min) {
    mid = (min + max) / 2;
    if (ucs > table[mid].last)
        min = mid + 1;
    else if (ucs < table[mid].first)
        max = mid - 1;
    else
        return 1;
}

return 0;

/* The following two functions define the column width of an ISO 10646 */

/* sorted list of non-overlapping intervals of non-spacing characters */
static const struct interval combining[] = {
    { 0x0300, 0x036F },
    { 0x0483, 0x0486 },
    { 0x0488, 0x0489 },
    { 0x0591, 0x05BD },
    { 0x05BF, 0x05BF },
    { 0x05C1, 0x05C2 },
    { 0x05C4, 0x05C5 },
    { 0x05C7, 0x05C7 },
    { 0x0600, 0x0603 },
    { 0x0610, 0x0615 },
    { 0x064B, 0x065E },
    { 0x0667, 0x0668 },
    { 0x066A, 0x066B },
    { 0x0670, 0x0670 },
    { 0x06E7, 0x06E8 },
    { 0x06EA, 0x06EC },
    { 0x070F, 0x070F },
    { 0x0711, 0x0711 },
    { 0x0730, 0x074A },
    { 0x07A6, 0x07B0 },
    { 0x07EB, 0x07F3 },
    { 0x0901, 0x0902 },
    { 0x093C, 0x093C },
    { 0x0941, 0x0948 },
    { 0x094D, 0x094D },
    { 0x0951, 0x0954 },
    { 0x0962, 0x0963 },
    { 0x0981, 0x0981 },
    { 0x099C, 0x099C },
    { 0x099D, 0x099D },
    { 0x09B1, 0x09B1 },
    { 0x09B2, 0x09B2 },
    { 0x09C1, 0x09C4 },
    { 0x09CD, 0x09CD },
    { 0x09E2, 0x09E2 },
    { 0x09E3, 0x09E3 },
    { 0x09ED, 0x09ED },
    { 0x09F7, 0x09F7 },
    { 0x09F8, 0x09F8 },
    { 0x09FA, 0x09FA },
    { 0x0A01, 0x0A02 },
    { 0x0A07, 0x0A07 },
    { 0x0A11, 0x0A11 },
    { 0x0A3C, 0x0A3C },
    { 0x0A41, 0x0A42 },
    { 0x0A47, 0x0A48 },
    { 0x0A4B, 0x0A4D },
    { 0x0B01, 0x0B01 },
    { 0x0B07, 0x0B07 },
    { 0x0B56, 0x0B56 },
    { 0x0B82, 0x0B82 },
    { 0x0B8C, 0x0B8C },
    { 0x0B90, 0x0B90 },
    { 0x0B91, 0x0B91 },
    { 0x0B9E, 0x0B9E },
    { 0x0BC0, 0x0BC0 },
    { 0x0BC1, 0x0BC1 },
    { 0x0BD2, 0x0BD2 },
    { 0x0C30, 0x0C30 },
    { 0x0C34, 0x0C34 },
    { 0x0C55, 0x0C56 },
    { 0x0CBC, 0x0CBC },
    { 0x0CD0, 0x0CD0 },
    { 0x0CD2, 0x0CD2 },
    { 0x0CD6, 0x0CD6 },
    { 0x0CCE, 0x0CCE },
    { 0x0CFC, 0x0CFC },
    { 0x0CF0, 0x0CF0 },
    { 0x0CF6, 0x0CF6 },
    { 0x0E33, 0x0E34 },
    { 0x0E3B, 0x0E3E },
    { 0x0E47, 0x0E4E },
    { 0x0E81, 0x0E81 },
    { 0x0EB4, 0x0EB5 },
    { 0x0EBB, 0x0EBC },
    { 0x0EC9, 0x0ECD },
    { 0x0F18, 0x0F19 },
    { 0x0F35, 0x0F35 },
    { 0x0F37, 0x0F37 },
    { 0x0F39, 0x0F39 },
    { 0x0F71, 0x0F7E },
    { 0x0F80, 0x0F84 },
    { 0x0F86, 0x0F87 },
    { 0x0F90, 0x0F97 },
    { 0x0F99, 0x0F9C },
    { 0x0FB6, 0x0FC6 },
    { 0x1039, 0x1039 },
    { 0x1058, 0x1059 },
    { 0x1160, 0x1160 },
    { 0x11E0, 0x11E0 },
    { 0x11F0, 0x11F0 },
    { 0x11FA, 0x11FA },
    { 0x11F9, 0x11F9 },
    { 0x135F, 0x135F },
    { 0x1732, 0x1732 },
    { 0x1734, 0x1734 },
    { 0x1752, 0x1753 },
};
13.300 mk_wcwidth.c

00180 0x1772, 0x1773 }, { 0x17B4, 0x17B5 }, { 0x17B7, 0x17BD },
00181 { 0x17C6, 0x17C6 }, { 0x17C9, 0x17D3 }, { 0x17DD, 0x17DD },
00182 { 0x180B, 0x180D }, { 0x18A9, 0x18A9 }, { 0x1920, 0x1922 },
00183 { 0x1927, 0x1928 }, { 0x1932, 0x1932 }, { 0x1939, 0x193B },
00184 { 0x1A17, 0x1A18 }, { 0x1B00, 0x1B03 }, { 0x1B34, 0x1B3A },
00185 { 0x1B36, 0x1B3B }, { 0x1DC0, 0x1DCA }, { 0x1DFE, 0x1DFF },
00186 { 0x200B, 0x200F }, { 0x202A, 0x202E }, { 0x2060, 0x2063 },
00187 { 0x206A, 0x206F }, { 0x20D0, 0x20EF }, { 0x302A, 0x302F },
00188 { 0x3099, 0x309A }, { 0xA806, 0xA806 }, { 0xA80B, 0xA80B },
00189 { 0xA825, 0xA826 }, { 0xFE00, 0xFE0F }, { 0xFE20, 0xFE23 },
00190 { 0xFEFF, 0xFEFF }, { 0x10A01, 0x10A03 }, { 0x10A05, 0x10A06 },
00191 { 0x10A0C, 0x10A0F }, { 0x1D167, 0x1D169 },
00192 { 0x1D173, 0x1D182 }, { 0x1D185, 0x1D18B }, { 0x1D1AA, 0x1D1AD },
00193 { 0x1D242, 0x1D244 }, { 0xFE001, 0xFE001 }, { 0xFE002, 0xFE007 },
00194 { 0xFE003, 0xFE019 },
00195 );
00196 */
00197 /* test for 8-bit control characters */
00198 if (ucs == 0)
00199 return 0;
00200 */
00201 if (ucs < 32 || (ucs >= 0x7f && ucs < 0xa0))
00202 return -1;
00203 */
00204 / binary search in table of non-spacing characters */
00205 if (bisearch(ucs, combining,
00206 sizeof(combining) / sizeof(struct interval) - 1))
00207 return 0;
00208 */
00209 if we arrive here, ucs is not a combining or C0/C1 control character */
00210 return 1 +
00211 /* FLTK: comment out the remaining functions, as we don't need them. */
00212 */
00213 int mk_wcwidth(const wchar_t *pwcs, size_t n)
00214 { int w, width = 0;
00215 for (; *pwcs && n-- > 0; pwcs++)
00216 if ((w = mk_wcwidth( *pwcs)) < 0)
00217 return -1;
00218 else
00219 width += w;
00220 return width;
00221 */
00222 int mk_wcwidth_cjk(unsigned int ucs)
00223 { int w, width = 0;
00224 for (; ucs && n-- > 0; ucs++)
00225 if ((w = mk_wcwidth_cjk(ucs)) < 0)
00226 return -1;
00227 else
00228 width += w;
00229 return width;
00230 */
00231 int mk_wcswidth(const wchar_t *pwcs, size_t n)
00232 { int w, width = 0;
00233 for (; *pwcs && n-- > 0; pwcs++)
00234 if ((w = mk_wcswidth(pwcs)) < 0)
00235 return -1;
00236 else
00237 width += w;
00238 return width;
00239 */
00240 */
00241 */
00242 */
00243 */
00244 */
00245 */
00246 */
00247 */
00248 */
00249 */
00250 */
00251 */
00252 */
00253 /* The following functions are the same as mk_wcswidth() and
00254 * mk_wcwidth(), except that spacing characters in the East Asian
00255 * ambiguous (A) category as defined in Unicode Technical Report #11
00256 * have a column width of 2. This variant might be useful for users of
00257 * CJK legacy encodings who want to migrate to UCS without changing
00258 * the traditional terminal character-width behaviour. It is not
00259 * otherwise recommended for general use.
00260 */
00261 */
00262 */
00263 int mk_wcswidth_cjk(wchar_t ucs)
00264 */
00265 int mk_wcswidth_cjk(unsigned int ucs)
00266 */

Generated by Doxygen
2644

00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353

File Documentation

/* sorted list of non-overlapping intervals of East Asian Ambiguous
* characters, generated by "uniset +WIDTH-A -cat=Me -cat=Mn -cat=Cf c" */
static const struct interval ambiguous[] = {
{ 0x00A1, 0x00A1 }, { 0x00A4, 0x00A4 }, { 0x00A7, 0x00A8 },
{ 0x00AA, 0x00AA }, { 0x00AE, 0x00AE }, { 0x00B0, 0x00B4 },
{ 0x00B6, 0x00BA }, { 0x00BC, 0x00BF }, { 0x00C6, 0x00C6 },
{ 0x00D0, 0x00D0 }, { 0x00D7, 0x00D8 }, { 0x00DE, 0x00E1 },
{ 0x00E6, 0x00E6 }, { 0x00E8, 0x00EA }, { 0x00EC, 0x00ED },
{ 0x00F0, 0x00F0 }, { 0x00F2, 0x00F3 }, { 0x00F7, 0x00FA },
{ 0x00FC, 0x00FC }, { 0x00FE, 0x00FE }, { 0x0101, 0x0101 },
{ 0x0111, 0x0111 }, { 0x0113, 0x0113 }, { 0x011B, 0x011B },
{ 0x0126, 0x0127 }, { 0x012B, 0x012B }, { 0x0131, 0x0133 },
{ 0x0138, 0x0138 }, { 0x013F, 0x0142 }, { 0x0144, 0x0144 },
{ 0x0148, 0x014B }, { 0x014D, 0x014D }, { 0x0152, 0x0153 },
{ 0x0166, 0x0167 }, { 0x016B, 0x016B }, { 0x01CE, 0x01CE },
{ 0x01D0, 0x01D0 }, { 0x01D2, 0x01D2 }, { 0x01D4, 0x01D4 },
{ 0x01D6, 0x01D6 }, { 0x01D8, 0x01D8 }, { 0x01DA, 0x01DA },
{ 0x01DC, 0x01DC }, { 0x0251, 0x0251 }, { 0x0261, 0x0261 },
{ 0x02C4, 0x02C4 }, { 0x02C7, 0x02C7 }, { 0x02C9, 0x02CB },
{ 0x02CD, 0x02CD }, { 0x02D0, 0x02D0 }, { 0x02D8, 0x02DB },
{ 0x02DD, 0x02DD }, { 0x02DF, 0x02DF }, { 0x0391, 0x03A1 },
{ 0x03A3, 0x03A9 }, { 0x03B1, 0x03C1 }, { 0x03C3, 0x03C9 },
{ 0x0401, 0x0401 }, { 0x0410, 0x044F }, { 0x0451, 0x0451 },
{ 0x2010, 0x2010 }, { 0x2013, 0x2016 }, { 0x2018, 0x2019 },
{ 0x201C, 0x201D }, { 0x2020, 0x2022 }, { 0x2024, 0x2027 },
{ 0x2030, 0x2030 }, { 0x2032, 0x2033 }, { 0x2035, 0x2035 },
{ 0x203B, 0x203B }, { 0x203E, 0x203E }, { 0x2074, 0x2074 },
{ 0x207F, 0x207F }, { 0x2081, 0x2084 }, { 0x20AC, 0x20AC },
{ 0x2103, 0x2103 }, { 0x2105, 0x2105 }, { 0x2109, 0x2109 },
{ 0x2113, 0x2113 }, { 0x2116, 0x2116 }, { 0x2121, 0x2122 },
{ 0x2126, 0x2126 }, { 0x212B, 0x212B }, { 0x2153, 0x2154 },
{ 0x215B, 0x215E }, { 0x2160, 0x216B }, { 0x2170, 0x2179 },
{ 0x2190, 0x2199 }, { 0x21B8, 0x21B9 }, { 0x21D2, 0x21D2 },
{ 0x21D4, 0x21D4 }, { 0x21E7, 0x21E7 }, { 0x2200, 0x2200 },
{ 0x2202, 0x2203 }, { 0x2207, 0x2208 }, { 0x220B, 0x220B },
{ 0x220F, 0x220F }, { 0x2211, 0x2211 }, { 0x2215, 0x2215 },
{ 0x221A, 0x221A }, { 0x221D, 0x2220 }, { 0x2223, 0x2223 },
{ 0x2225, 0x2225 }, { 0x2227, 0x222C }, { 0x222E, 0x222E },
{ 0x2234, 0x2237 }, { 0x223C, 0x223D }, { 0x2248, 0x2248 },
{ 0x224C, 0x224C }, { 0x2252, 0x2252 }, { 0x2260, 0x2261 },
{ 0x2264, 0x2267 }, { 0x226A, 0x226B }, { 0x226E, 0x226F },
{ 0x2282, 0x2283 }, { 0x2286, 0x2287 }, { 0x2295, 0x2295 },
{ 0x2299, 0x2299 }, { 0x22A5, 0x22A5 }, { 0x22BF, 0x22BF },
{ 0x2312, 0x2312 }, { 0x2460, 0x24E9 }, { 0x24EB, 0x254B },
{ 0x2550, 0x2573 }, { 0x2580, 0x258F }, { 0x2592, 0x2595 },
{ 0x25A0, 0x25A1 }, { 0x25A3, 0x25A9 }, { 0x25B2, 0x25B3 },
{ 0x25B6, 0x25B7 }, { 0x25BC, 0x25BD }, { 0x25C0, 0x25C1 },
{ 0x25C6, 0x25C8 }, { 0x25CB, 0x25CB }, { 0x25CE, 0x25D1 },
{ 0x25E2, 0x25E5 }, { 0x25EF, 0x25EF }, { 0x2605, 0x2606 },
{ 0x2609, 0x2609 }, { 0x260E, 0x260F }, { 0x2614, 0x2615 },
{ 0x261C, 0x261C }, { 0x261E, 0x261E }, { 0x2640, 0x2640 },
{ 0x2642, 0x2642 }, { 0x2660, 0x2661 }, { 0x2663, 0x2665 },
{ 0x2667, 0x266A }, { 0x266C, 0x266D }, { 0x266F, 0x266F },
{ 0x273D, 0x273D }, { 0x2776, 0x277F }, { 0xE000, 0xF8FF },
{ 0xFFFD, 0xFFFD }, { 0xF0000, 0xFFFFD }, { 0x100000, 0x10FFFD }
};
/* binary search in table of non-spacing characters */
if (bisearch(ucs, ambiguous,
sizeof(ambiguous) / sizeof(struct interval) - 1))
return 2;
return mk_wcwidth(ucs);
}

/*
* FLTK: was
int mk_wcswidth_cjk(const wchar_t *pwcs, size_t n)
*/
int mk_wcswidth_cjk(const unsigned int *pwcs, size_t n)
{
int w, width = 0;
for (;*pwcs && n-- > 0; pwcs++)
if ((w = mk_wcwidth_cjk(*pwcs)) < 0)
return -1;
else
width += w;
return width;
}
/*
* FLTK: end of commented out functions
*/
#endif

Generated by Doxygen


/*
 * Author: Jean-Marc Lienher (http://oksid.ch)
 * Copyright 2000-2003 by O'ksi'D.
 *
 * This library is free software. Distribution and use rights are outlined in
 * the file "COPYING" which should have been included with this file. If this
 * file is missing or damaged, see the license at:
 *
 * https://www.fltk.org/COPYING.php
 *
 * Please see the following page on how to report bugs and issues:
 *
 * https://www.fltk.org/bugs.php
 */

#include <stdlib.h>
#include <string.h>

#define RET_ILSEQ -1
#define RET_TOOFEW(x) (-10 - x)
#define RET_TOOSMALL -2
#define conv_t void*
#define ucs4_t unsigned int

typedef struct {
  unsigned short indx;
  unsigned short used;
} Summary16;

NEED_TOMB /* indicates what part of these include files is needed here (avoid compilation
  warnings) */

#include "lcUniConv/cp936ext.h"
#include "lcUniConv/big5.h"
#include "lcUniConv/gb2312.h"
#include "lcUniConv/iso8859_10.h"
#include "lcUniConv/iso8859_11.h"
#include "lcUniConv/iso8859_13.h"
#include "lcUniConv/iso8859_14.h"
#include "lcUniConv/iso8859_15.h"
#include "lcUniConv/iso8859_2.h"
#include "lcUniConv/iso8859_3.h"
#include "lcUniConv/iso8859_4.h"
#include "lcUniConv/iso8859_5.h"
#include "lcUniConv/iso8859_6.h"
#include "lcUniConv/iso8859_7.h"
#include "lcUniConv/iso8859_8.h"
#include "lcUniConv/iso8859_9.h"
#include "lcUniConv/jisx0201.h"
#include "lcUniConv/jisx0208.h"
#include "lcUniConv/jisx0212.h"
#include "lcUniConv/koi8_r.h"
#include "lcUniConv/koi8_u.h"
#include "lcUniConv/ksc5601.h"
#include "lcUniConv/cp1251.h"
#include "headers/symbol_.h"
#include "headers/dingbats_.h"

/******* conv_gen.c ************/

const static int ucs2fontmap(char *s, unsigned int ucs, int enc) {
  switch(enc) {
  case 0: /* iso10646-1 */
    s[0] = (char) ((ucs & 0xFF00) >> 8);
    s[1] = (char) (ucs & 0xFF);
    return 0;
  case 1: /* iso8859-1 */
    if (ucs <= 0x00FF) {
      if (ucs >= 0x0001) {
        s[0] = 0;
        s[1] = (char) (ucs & 0xFF);
        return 1;
      }
    }
  case 2: /* iso8859-2 */
    if (ucs <= 0x00FF) {
      if (ucs >= 0x0001) {
        s[0] = 0;
        s[1] = (char) (ucs & 0xFF);
        return 1;
      }
    }
    if (s[1]) return 2;
    return 1;
  case 3: /* iso8859-3 */
    if (s[1]) return 2;
  case 4: /* iso8859-4 */
    if (s[1]) return 2;
  case 5: /* iso8859-5 */
    if (s[1]) return 2;
  case 6: /* iso8859-6 */
    if (s[1]) return 2;
  case 7: /* iso8859-7 */
    if (s[1]) return 2;
  case 8: /* iso8859-8 */
    if (s[1]) return 2;
  case 9: /* iso8859-9 */
    if (s[1]) return 2;
  case 10: /* jisx0201 */
    if (s[1]) return 2;
  case 11: /* jisx0208 */
    if (s[1]) return 2;
  case 12: /* jisx0212 */
    if (s[1]) return 2;
  case 13: /* koi8_r */
    if (s[1]) return 2;
  case 14: /* koi8_u */
    if (s[1]) return 2;
  case 15: /* ksc5601 */
    if (s[1]) return 2;
  case 16: /* cp1251 */
    if (s[1]) return 2;
  case 17: /* symbol_ */
    if (s[1]) return 2;
  case 18: /* dingbats_ */
    if (s[1]) return 2;
  default:
    break;
  }
  switch(enc) {
  case 0: /* iso10646-1 */
    if (ucs <= 0x00FF) {
      if (ucs >= 0x0001) {
        s[0] = 0;
        s[1] = (char) (ucs & 0xFF);
        return 1;
      }
    }
    if (s[1]) return 2;
    return 1;
  case 1: /* iso8859-1 */
    if (s[1]) return 2;
  case 2: /* iso8859-2 */
    if (s[1]) return 2;
  case 3: /* iso8859-3 */
    if (s[1]) return 2;
  case 4: /* iso8859-4 */
    if (s[1]) return 2;
  case 5: /* iso8859-5 */
    if (s[1]) return 2;
  case 6: /* iso8859-6 */
    if (s[1]) return 2;
  case 7: /* iso8859-7 */
    if (s[1]) return 2;
  case 8: /* iso8859-8 */
    if (s[1]) return 2;
  case 9: /* iso8859-9 */
    if (s[1]) return 2;
  case 10: /* jisx0201 */
    if (s[1]) return 2;
  case 11: /* jisx0208 */
    if (s[1]) return 2;
  case 12: /* jisx0212 */
    if (s[1]) return 2;
  case 13: /* koi8_r */
    if (s[1]) return 2;
  case 14: /* koi8_u */
    if (s[1]) return 2;
  case 15: /* ksc5601 */
    if (s[1]) return 2;
  case 16: /* cp1251 */
    if (s[1]) return 2;
  case 17: /* symbol_ */
    if (s[1]) return 2;
  case 18: /* dingbats_ */
    if (s[1]) return 2;
  default:
    break;
  }
  return 1;
}

Generated by Doxygen
break;
}
00173 case 18: /* jisx0201.1976-0 */
00174 if (jisx0201_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
00175    return 18;
00176 }
00177 break;
00178 case 19: /* jisx0208.1983-0 */
00179 if (jisx0208_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
00180    return 19;
00181 }
00182 break;
00183 case 20: /* jisx0212.1990-0 */
00184 if (jisx0212_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
00185    return 20;
00186 }
00187 break;
00188 case 21: /* symbol */
00189 if (ucs <= 0x0080) {
00190    if (ucs >= 0x0020) {
00191        s[0] = 0;
00192        s[1] = unicode_to_symbol_1b_0020[ucs - 0x0020];
00193        if (s[1]) return 21;
00194    }
00195 } else if (ucs <= 0x0192) {
00196    if (ucs >= 0x0192) {
00197        s[0] = 0;
00198        s[1] = unicode_to_symbol_1b_0192[ucs - 0x0192];
00199        if (s[1]) return 21;
00200    }
00201 } else if (ucs <= 0x232A) {
00202    if (ucs >= 0x0202) {
00203        s[0] = 0;
00204        s[1] = unicode_to_symbol_1b_2022[ucs - 0x2022];
00205        if (s[1]) return 21;
00206    }
00207 } else if (ucs <= 0x2666) {
00208    if (ucs >= 0x2660) {
00209        s[0] = 0;
00210        s[1] = unicode_to_symbol_1b_2660[ucs - 0x2660];
00211        if (s[1]) return 21;
00212    }
00213 } else if (ucs <= 0x2FDF) {
00214    if (ucs >= 0x2F69) {
00215        s[0] = 0;
00216        s[1] = unicode_to_symbol_1b_2F69[ucs - 0x2F69];
00217        if (s[1]) return 21;
00218    }
00219 } else if (ucs <= 0x300F) {
00220    if (ucs >= 0x304D) {
00221        s[0] = 0;
00222        s[1] = unicode_to_symbol_1b_304D[ucs - 0x304D];
00223        if (s[1]) return 21;
00224    }
00225 } else if (ucs <= 0x3CDE) {
00226    if (ucs >= 0x3C28) {
00227        s[0] = 0;
00228        s[1] = unicode_to_symbol_1b_3C28[ucs - 0x3C28];
00229        if (s[1]) return 21;
00230    }
00231 } else if (ucs <= 0x4E81) {
00232    if (ucs >= 0x4E00) {
00233        s[0] = 0;
00234        s[1] = unicode_to_symbol_1b_4E00[ucs - 0x4E00];
00235        if (s[1]) return 21;
00236    }
00237 }
00238 break;
00239 case 22: /* dingbats */
00240 if (ucs <= 0x00A0) {
00241    if (ucs >= 0x0020) {
00242        s[0] = 0;
00243        s[1] = unicode_to_dingbats_1b_0020[ucs - 0x0020];
00244        if (s[1]) return 22;
00245    }
00246 } else if (ucs <= 0x2195) {
00247    if (ucs >= 0x2192) {
00248        s[0] = 0;
00249        s[1] = unicode_to_dingbats_1b_2192[ucs - 0x2192];
00250        if (s[1]) return 22;
00251    }
00252 } else if (ucs <= 0x2469) {
00253    if (ucs >= 0x2460) {
00254        s[0] = 0;
00255        s[1] = unicode_to_dingbats_1b_2460[ucs - 0x2460];
00256        if (s[1]) return 22;
00257    }
00258 } else if (ucs <= 0x2666) {

Generated by Doxygen
if (ucs >= 0x25A0) {
    s[0] = 0;
    s[1] = unicode_to_dingbats_1b_25A0[ucs - 0x25A0];
    if (s[1]) return 22;
} else if (ucs <= 0x27BE) {
    if (ucs >= 0x2701) {
        s[0] = 0;
        s[1] = unicode_to_dingbats_1b_2701[ucs - 0x2701];
        if (s[1]) return 22;
    }
    else if (ucs <= 0xF8E4) {
        if (ucs >= 0xF8D7) {
            s[0] = 0;
            s[1] = unicode_to_dingbats_1b_F8D7[ucs - 0xF8D7];
            if (s[1]) return 22;
        }
        break;
    }
    case 23: /* koi8-u */
    if (koi8_u_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
        return 23;
    }
    break;
    case 24: /* microsoft-cp1251 */
    if (cp1251_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
        return 24;
    }
    break;
    case 26: /* gbk/cp936ext */
    if (cp936ext_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
        return 26;
    }
    default:
        break;
    return -1;
}

static int encoding_number(const char *enc) {
    if (!enc || !strncmp(enc, "iso10646-1", 10)) {
        return 0;
    } else if (!strcmp(enc, "iso8859-1")) {
        return 1;
    } else if (!strcmp(enc, "iso8859-2")) {
        return 2;
    } else if (!strcmp(enc, "iso8859-3")) {
        return 3;
    } else if (!strcmp(enc, "iso8859-4")) {
        return 4;
    } else if (!strcmp(enc, "iso8859-5")) {
        return 5;
    } else if (!strcmp(enc, "iso8859-6")) {
        return 6;
    } else if (!strcmp(enc, "iso8859-7")) {
        return 7;
    } else if (!strcmp(enc, "iso8859-8")) {
        return 8;
    } else if (!strcmp(enc, "iso8859-9")) {
        return 9;
    } else if (!strcmp(enc, "iso8859-10")) {
        return 10;
    } else if (!strcmp(enc, "iso8859-13")) {
        return 11;
    } else if (!strcmp(enc, "iso8859-14")) {
        return 12;
    } else if (!strcmp(enc, "iso8859-15")) {
        return 13;
    } else if (!strcmp(enc, "koi8-r")) {
        return 14;
    } else if (!strcmp(enc, "big5-0") || !strcmp(enc, "big5.eten-0") ||
        !strcmp(enc, "big5-0");
        return 15;
    } else if (!strcmp(enc, "ksc5601.1987-0")) {
        return 16;
    } else if (!strcmp(enc, "gb2312.1980-0") || !strcmp(enc, "gb2312.80-0") ||
        !strcmp(enc, "gb2312.80gb565.88") || !strcmp(enc, "gb2312.80-0")) {
        return 17;
    } else if (!strcmp(enc, "jisx0201.1976-0") ||
        return 18;
    } else if (!strcmp(enc, "jisx0208.1983-0") ||
        !strcmp(enc, "jisx0208.1990-0") ||
        !strcmp(enc, "jisx0208.1978-0")) {
        return 19;
    }
```c
00346     return 19;
00347 } else if (!strcmp(enc, "jisx0212.1990-0")) {
00348     return 20;
00349 } else if (!strcmp(enc, "symbol")) {
00350     return 21;
00351 } else if (!strcmp(enc, "dingbats") || !strcmp(enc, "zapfdingbats") ||
00352     !strcmp(enc, "itc zapf dingbats")) {
00353     return 22;
00354 } else if (!strcmp(enc, "koi8-u")) {
00355     return 23;
00356 } else if (!strcmp(enc, "iso8859-11")) {
00357     return 24;
00358 } else if (!strcmp(enc, "cp936") || !strcmp(enc, "gbk") ||
00359     !strcmp(enc, "gbk-0") || !strcmp(enc, "cp1251") || !strcmp(enc, "gbk-0") ||
00360     !strcmp(enc, "gbk-0") || !strcmp(enc, "gbk-0") || !strcmp(enc, "gbk-0") ||
00361     !strcmp(enc, "gbk-0") || !strcmp(enc, "gbk-0") || !strcmp(enc, "gbk-0") ||
00362     !strcmp(enc, "gbk-0") || !strcmp(enc, "gbk-0") || !strcmp(enc, "gbk-0") ||
00363     !strcmp(enc, "gbk-0") ||
00364     return -1;
00365 }

13.302 utf8Utils.c

00001 */
00002 */ Author: Jean-Marc Lienher { http://oksid.ch }
00003 */ Copyright 2000-2003 by O'ksi'D.
00004 */
00005 */ This library is free software. Distribution and use rights are outlined in
00006 */ the file "COPYING" which should have been included with this file. If this
00007 */ file is missing or damaged, see the license at:
00008 */
00009 */   https://www.fltk.org/COPYING.php
00010 */
00011 */ Please see the following page on how to report bugs and issues:
00012 */
00013 */   https://www.fltk.org/bugs.php
00014 */
00015 */
00016 */ Unicode to UTF-8 conversion functions.
00017 */
00018 */ This file is compiled and linked only for X11 w/o Xft.
00019 */
00020 */
00021 */
00022 */ include "../Xutf8.h"
00023 */
00024 /*** NOTE : all functions are LIMITED to 24 bits Unicode values !!! ***/
00025 */
00026 */
00027 */ Converts the first char of the UTF-8 string to an Unicode value
00028 */
00029 */ Returns the byte length of the converted UTF-8 char
00030 */
00031 int
00032 XConvertUtf8ToUcs(const unsigned char *buf,
00033 int len,
00034 unsigned int *ucs) { ...
```
```c
/* 0x00010000 - 0x001FFFFF */
ucs = ((buf[0] & ~0xF0) « 18) +
((buf[1] & ~0x80) « 12) +
((buf[2] & ~0x80) « 6) +
(buf[3] & ~0x80);
if (ucs > 0x0000FFFF) return 4;

/* 0x00000800 - 0x0000FFFF */
ucs = ((buf[0] & ~0xE0) « 12) +
((buf[1] & ~0x80) « 6) +
(buf[2] & ~0x80);
if (ucs > 0x000007FF) return 3;

/* 0x00000080 - 0x000007FF */
ucs = ((buf[0] & ~0xC0) « 6) +
(buf[1] & ~0x80);
if (ucs > 0x0000007F) return 2;

/* 0x00000000 - 0x0000007F */
ucs = buf[0];
return 1;

/* bad UTF-8 string */
return -1;

int XConvertUcsToUtf8(unsigned int ucs,
char *buf) {
if (ucs < 0x000080) {
    buf[0] = ucs;
    return 1;
} else if (ucs < 0x000800) {
    buf[0] = 0xC0 | (ucs » 6);
    buf[1] = 0x80 | (ucs & 0x3F);
    return 2;
} else if (ucs < 0x010000) {
    buf[0] = 0xE0 | (ucs » 12);
    buf[1] = 0x80 | ((ucs » 6) & 0x3F);
    buf[2] = 0x80 | (ucs & 0x3F);
    return 3;
} else if (ucs < 0x00200000) {
    buf[0] = 0xF0 | (ucs » 18);
    buf[1] = 0x80 | ((ucs » 12) & 0x3F);
    buf[2] = 0x80 | ((ucs » 6) & 0x3F);
    buf[3] = 0x80 | (ucs & 0x3F);
    return 4;
} else if (ucs < 0x01000000) {
    buf[0] = 0xF8 | (ucs » 24);
    return 5;
}

buf[0] = '?';
return -1;

int XUtf8CharByteLen(const unsigned char *buf,
int len) {
unsigned int ucs;
return XConvertUtf8ToUcs(buf, len, &ucs);
}
```
XCountUtf8Char

```c
int XCountUtf8Char(const unsigned char *buf, int len)
{
    int i = 0;
    int nbc = 0;
    while (i < len) {
        int cl = XUtf8CharByteLen(buf + i, len - i);
        if (cl < 1) cl = 1;
        nbc++;
        i += cl;
    }
    return nbc;
}
```

Same as XConvertUtf8ToUcs but no sanity check is done.

```c
int XFastConvertUtf8ToUcs(const unsigned char *buf, int len, unsigned int *ucs)
{
    if (buf[0] & 0x80) {
        if (buf[0] & 0x40) {
            if (buf[0] & 0x20) {
                if (buf[0] & 0x10) {
                    if (buf[0] & 0x08) {
                        if (buf[0] & 0x04) {
                            if (buf[0] & 0x02) {
                                /* bad UTF-8 string */
                            } else {
                                /* 0x04000000 - 0x7FFFFFFF */
                            }
                        } else if (len > 4) {
                            /* 0x03000000 - 0x7fffffff */
                            *ucs = ((buf[0] & ~0xF8) << 24) +
                                    ((buf[1] & ~0x80) << 18) +
                                    ((buf[2] & ~0x80) << 12) +
                                    ((buf[3] & ~0x80) << 6) +
                                    (buf[4] & ~0x80);
                            return 5;
                        } else if (len > 3) {
                            /* 0x02000000 - 0x3fffffff */
                            *ucs = ((buf[0] & ~0xF0) << 18) +
                                    ((buf[1] & ~0x80) << 12) +
                                    ((buf[2] & ~0x80) << 6) +
                                    (buf[3] & ~0x80);
                            return 4;
                        } else if (len > 2) {
                            /* 0x01000000 - 0x2fffffff */
                            *ucs = ((buf[0] & ~0xE0) << 12) +
                                    ((buf[1] & ~0x80) << 6) +
                                    (buf[2] & ~0x80);
                            return 3;
                        } else if (len > 1) {
                            /* 0x00800000 - 0x1fffffff */
                            *ucs = (buf[0] & ~0xC0) << 6 +
                                    (buf[1] & ~0x80);
                            return 2;
                        } else if (len > 0) {
                            /* 0x00000000 - 0x7fff */
                            *ucs = buf[0];
                            return 1;
                        } else {
                            /* 0x00000000 - 0x0000007f */
                            *ucs = (unsigned int) '?'; /* bad UTF-8 string */
                            return -1;
                        }
                    } else if (buf[0] & 0x40) {
                        /* 0x00800000 - 0x007FFFFF */
                        *ucs = (buf[0] & ~0x80);
                        return 1;
                    } else if (buf[0] & 0x20) {
                        /* 0x00000000 - 0x0000007f */
                        *ucs = (unsigned int) '?'; /* bad UTF-8 string */
                        return -1;
                    }
                } else if (len > 3) {
                    /* 0x00010000 - 0x001fffff */
                    *ucs = ((buf[0] & ~0xF0) << 18) +
                            ((buf[1] & ~0x80) << 12) +
                            ((buf[2] & ~0x80) << 6) +
                            (buf[3] & ~0x80);
                    return 4;
                } else if (len > 2) {
                    /* 0x00000800 - 0x000007ff */
                    *ucs = ((buf[0] & ~0xE0) << 12) +
                            ((buf[1] & ~0x80) << 6) +
                            (buf[2] & ~0x80);
                    return 3;
                } else if (len > 1) {
                    /* 0x00000080 - 0x0000007f */
                    *ucs = (buf[0] & ~0xC0) << 6 +
                            (buf[1] & ~0x80);
                    return 2;
                } else if (len > 0) {
                    /* 0x00000000 - 0x0000000f */
                    *ucs = buf[0];
                    return 1;
                } else {
                    /* 0x00000000 - 0x00000001 */
                    *ucs = (unsigned int) '1'; /* bad UTF-8 string */
                    return -1;
                }
            } else if (len > 0) {
                /* 0x00000000 - 0x00000001 */
                *ucs = (unsigned int) '1'; /* bad UTF-8 string */
                return -1;
            }
        } else if (len > 0) {
            /* 0x00000000 - 0x00000001 */
            *ucs = (unsigned int) '1'; /* bad UTF-8 string */
            return -1;
        }
    } else if (len > 0) {
        /* 0x00000000 - 0x00000001 */
        *ucs = (unsigned int) '1'; /* bad UTF-8 string */
        return -1;
    }
```
Index

_\_FL\_DIAMOND\_DOWN\_BOX
Enumerations.H, 1980

_\_FL\_DIAMOND\_UP\_BOX
Enumerations.H, 1980

_\_FL\_EMBOSSSED\_LABEL
Enumerations.H, 1986

_\_FL\_ENGRAVED\_LABEL
Enumerations.H, 1986

_\_FL\_GLEAM\_DOWN\_BOX
Enumerations.H, 1980

_\_FL\_GLEAM\_DOWN\_FRAME
Enumerations.H, 1981

_\_FL\_GLEAM\_ROUND\_DOWN\_BOX
Enumerations.H, 1981

_\_FL\_GLEAM\_ROUND\_UP\_BOX
Enumerations.H, 1981

_\_FL\_GLEAM\_THIN\_DOWN\_BOX
Enumerations.H, 1981

_\_FL\_GLEAM\_THIN\_UP\_BOX
Enumerations.H, 1981

_\_FL\_GLEAM\_UP\_BOX
Enumerations.H, 1980

_\_FL\_GLEAM\_UP\_FRAME
Enumerations.H, 1981

_\_FL\_GTK\_DOWN\_BOX
Enumerations.H, 1980

_\_FL\_GTK\_DOWN\_FRAME
Enumerations.H, 1980

_\_FL\_GTK\_ROUND\_DOWN\_BOX
Enumerations.H, 1980

_\_FL\_GTK\_ROUND\_UP\_BOX
Enumerations.H, 1980

_\_FL\_GTK\_THIN\_DOWN\_BOX
Enumerations.H, 1980

_\_FL\_GTK\_THIN\_DOWN\_FRAME
Enumerations.H, 1980

_\_FL\_GTK\_THIN\_UP\_BOX
Enumerations.H, 1980

_\_FL\_GTK\_THIN\_UP\_FRAME
Enumerations.H, 1980

_\_FL\_GTK\_UP\_BOX
Enumerations.H, 1980

_\_FL\_GTK\_UP\_FRAME
Enumerations.H, 1980

_\_FL\_ICON\_LABEL
Enumerations.H, 1986

_\_FL\_IMAGE\_LABEL
Enumerations.H, 1986

_\_FL\_MULTI\_LABEL
Enumerations.H, 1986

_\_FL\_OFLAT\_BOX
Enumerations.H, 1986

_\_FL\_OSHADOW\_BOX
Enumerations.H, 1980

_\_FL\_OVAL\_BOX
Enumerations.H, 1980

_\_FL\_OVAL\_FRAME
Enumerations.H, 1980

_\_FL\_OXY\_BUTTON\_DOWN\_BOX
Enumerations.H, 1981

_\_FL\_OXY\_BUTTON\_UP\_BOX
Enumerations.H, 1981

_\_FL\_OXY\_DOWN\_BOX
Enumerations.H, 1981

_\_FL\_OXY\_DOWN\_FRAME
Enumerations.H, 1981

_\_FL\_OXY\_ROUND\_DOWN\_BOX
Enumerations.H, 1981

_\_FL\_OXY\_ROUND\_UP\_BOX
Enumerations.H, 1981

_\_FL\_OXY\_THIN\_DOWN\_BOX
Enumerations.H, 1981

_\_FL\_OXY\_THIN\_DOWN\_FRAME
Enumerations.H, 1981

_\_FL\_OXY\_THIN\_UP\_BOX
Enumerations.H, 1981

_\_FL\_OXY\_THIN\_UP\_FRAME
Enumerations.H, 1981

_\_FL\_OXY\_UP\_BOX
Enumerations.H, 1981

_\_FL\_OXY\_UP\_FRAME
Enumerations.H, 1981

_\_FL\_PLASTIC\_DOWN\_BOX
Enumerations.H, 1980

_\_FL\_PLASTIC\_DOWN\_FRAME
Enumerations.H, 1980

_\_FL\_PLASTIC\_ROUND\_DOWN\_BOX
Enumerations.H, 1980

_\_FL\_PLASTIC\_ROUND\_UP\_BOX
Enumerations.H, 1980

_\_FL\_PLASTIC\_THIN\_DOWN\_BOX
Enumerations.H, 1980

_\_FL\_PLASTIC\_THIN\_UP\_BOX
Enumerations.H, 1980

_\_FL\_PLASTIC\_UP\_BOX
Enumerations.H, 1980

_\_FL\_PLASTIC\_UP\_FRAME
Enumerations.H, 1980

Generated by Doxygen
add_extra
  Fl_File_Chooser, 688
add_fd
  Fl, 401
add_handler
  Events handling functions, 271
  Fl_Shared_Image, 1406
add_idle
  Fl, 401
add_key_binding
  Fl_Text_Editor, 1686
add_modify_callback
  Fl_Text_Buffer, 1610
add_scheme_name
  Fl_Scheme, 1340
add_system_handler
  Events handling functions, 271
add_timeout
  Fl, 401
  Fl_Timeout, 1719
add_vertex
  Fl_File_Icon, 691, 692
Adding and Extending Widgets, 81
addPlugin
  Fl_Plugin_Manager, 1202
address
  Fl_Text_Buffer, 1610
Advanced FLTK, 135
align
  Fl_Widget, 1874, 1875
alloc_size
  Fl_Flex, 735
ALWAYS_ON
  Fl_Browser_, 505
angle1
  Fl_Dial, 648
animate
  Fl_GIF_Image, 784
ansi
  Fl_Terminal, 1588
api_version
  Fl, 402
append
  Fl_Input_, 970
  Fl_Terminal, 1588
  Fl_Text_Buffer, 1610
append_ascii
  Fl_Terminal, 1588
append_utf8
  Fl_Text_Buffer, 1611
apply_undo
  Fl_Input_, 970
arg
  Fl, 402
args
  Fl, 403
args_to_utf8
  Fl, 404
argument
  Fl_Menu_Item, 1081
  Fl_Widget, 1875
armsci_8.h, 2362
array
  Fl_Group, 853
  Fl_RGB_Image, 1314
  Fl_Table, 1518
arrow_widths
  Fl.Counter, 636
as_double_window
  Fl_Double_Window, 662
  Fl_Window, 1923
as_gl_window
  Fl_Gl_Window, 799
  Fl_Widget, 1875
as_group
  Fl_Group, 853
  Fl_Widget, 1876
as_overlay_window
  Fl_Overlay_Window, 1175
  Fl_Window, 1923
as_shared_image
  Fl_Image, 934
  Fl_Shared_Image, 1406
as_svg_image
  Fl_RGB_Image, 1312
  Fl_SVG_Image, 1486
as_window
  Fl_Widget, 1876
  Fl_Window, 1923
ascii.h, 2363
atclose
  Windows handling functions, 268
ATTR_BGCOLOR
  Fl_Text_Display, 1639
ATTR_BGCOLOR_EXT
  Fl_Text_Display, 1639
ATTR_BGCOLOR_EXT_
  Fl_Text_Display, 1639
ATTR_GRAMMAR
  Fl_Text_Display, 1639
ATTR_LINES_MASK
  Fl_Text_Display, 1640
ATTR_SPELLING
  Fl_Text_Display, 1640
ATTRStrike_THROUGH
  Fl_Text_Display, 1640
ATTR_UNDERLINE
  Fl_Text_Display, 1639
Attrib
  Fl_Terminal, 1585
AUTO_DELETE_USER_DATA
  Fl_Widget, 1873
autosize
  Fl_Chart, 553
awake
  Multithreading support functions, 339
b
  Fl_Color_Chooser, 621
  Fl_Rect, 1289
B0
  Fl_Paged_Device, 1189
B1
  Fl_Paged_Device, 1189
B10
  Fl_Paged_Device, 1189
B2
  Fl_Paged_Device, 1189
B3
  Fl_Paged_Device, 1189
B4
  Fl_Paged_Device, 1189
B5
  Fl_Paged_Device, 1189
B6
  Fl_Paged_Device, 1189
B7
  Fl_Paged_Device, 1189
B8
  Fl_Paged_Device, 1189
B9
  Fl_Paged_Device, 1189
background
  Fl, 405
background2
  Fl, 405
bbox
  Fl_Browser_, 506
  Fl_Scroll, 1362
begin
  Fl_Group, 853
begin_job
  Fl_Paged_Device, 1190
  Fl_PostScript_File_Device, 1223, 1224
  Fl_Printer, 1253
begin_page
  Fl_Paged_Device, 1190
  Fl_PostScript_File_Device, 1224
  Fl_Printer, 1253
belowmouse
  Events handling functions, 271, 272
BG_XTERM
  Fl_Terminal, 1586
big5.h, 2364
big5_emacs.h, 2411
bind_deimage
  Fl_Widget, 1877
bind_image
  Fl_Widget, 1877, 1878
BLOCK_CURSOR
  Fl_Text_Display, 1639
BOLD
  Fl_Terminal, 1585
border
  Fl_Window, 1924
BOTH
  Fl_Browser_, 505
BOTH_ALWAYS
  Fl_Browser_, 505
bottomline
  Fl_Browser, 476
bounds
  Fl_Chart, 553
  Fl_Group, 853
  Fl_Slider, 1450
box
  Fl_Terminal, 1589
  Fl_Widget, 1878
box_border_radius_max
  Fl, 405
box_color
  Fl, 406
box_dh
  Fl, 406
box_dw
  Fl, 406
box_dx
  Fl, 406
box_dy
  Fl, 406
box_shadow_width
  Fl, 407
BROWSE_DIRECTORY
  Fl_Native_File_Chooser, 1140
BROWSE_FILE
  Fl_Native_File_Chooser, 1140
BROWSE_MULTI_DIRECTORY
  Fl_Native_File_Chooser, 1140
BROWSE_MULTI_FILE
  Fl_Native_File_Chooser, 1140
BROWSE_SAVE_DIRECTORY
  Fl_Native_File_Chooser, 1140
BROWSE_SAVE_FILE
  Fl_Native_File_Chooser, 1140
buffer
  Fl_Text_Display, 1640, 1641
buffer_modified_cb
  Fl_Text_Display, 1641
buffer_predelete_cb
  Fl_Text_Display, 1642
byte_at
  Fl_Text_Buffer, 1611
C_LOCALE
  Fl_Preferences, 1233
Cairo Support Functions and Classes, 342
cairo_autolink_context, 342
cairo_cc, 343
cairo_flush, 344
cairo_make_current, 344
cairo_autolink_context
  Cairo Support Functions and Classes, 343
compact
   Fl_Button, 526
compare
   Fl_Shared_Image, 1407
compose
   Events handling functions, 272
compose_reset
   Events handling functions, 273
configure_edges
   gtk_surface1_listener, 1956
connectorstyle
   Fl_Tree, 1767
Constants and Enumerations, 148
contains
   Fl_Widget, 1882
copy
   Fl_Anim_GIF_Image, 437
   Fl_Bitmap, 448
   Fl_Image, 935
   Fl_Input_, 971
   Fl_Menu_, 1045
   Fl_Pixmap, 1196
   Fl_RGB_Image, 1312
   Fl_Shared_Image, 1407, 1408
   Fl_SVG_Image, 1486
   Fl_Text_Buffer, 1612
   Fl_Tiled_Image, 1716
   Selection & Clipboard functions, 284
copy_
   Fl_Shared_Image, 1408
copy_cuts
   Fl_Input_, 971
copy_label
   Fl_Widget, 1882
copy_tooltip
   Fl_Widget, 1883
CORE
   Fl_Preferences, 1233
CORE_READ_OK
   Fl_Preferences, 1248
CORE_SYSTEM
   Fl_Preferences, 1233
CORE_SYSTEM_L
   Fl_Preferences, 1233
CORE_USER
   Fl_Preferences, 1233
CORE_USER_L
   Fl_Preferences, 1233
CORE_WRITE_OK
   Fl_Preferences, 1248
counter
   Fl_Image, 936
   Fl_Native_File_Chooser, 1141
count_displayed_characters
   Fl_Text_Buffer, 1612
count_lines
   Fl_Text_Buffer, 1612
   Fl_Text_Display, 1643
cp1133.h, 2413
cp1251.h, 2414
cp1255.h, 2416
cp1256.h, 2417
cp936ext.h, 2419
CR_TO_LF
   Fl_Terminal, 1586
CREATE
   Fl_File_Chooser, 687
create_window_menu
   Fl_Sys_Menu_Bar, 1500
current
   Fl_Group, 854, 855
   Fl_Timeout, 1720
   Fl_Tooltip, 1741
   Fl_Window, 1924
current_
   Fl_Window, 1940
current_timeout
   Fl_Timeout, 1723
cursor
   Fl_Tile, 1707
   Fl_Window, 1924, 1925
cursor2rowcol
   Fl_Table, 1521
cursor_col
   Fl_Widget, 1882
cursor_color
   Fl_Input_, 971
   Fl_Text_Display, 1643
   Fl_Value_Input, 1842
cursor_cr
   Fl_Terminal, 1590
Generated by Doxygen
cursor_down
    FI_Terminal, 1590
cursor_right
    FI_Terminal, 1591
cursor_row
    FI_Terminal, 1591
cursor_style
    FI_Text_Display, 1644
cursor_up
    FI_Terminal, 1591
custom_application_menu_items
    FI_Mac_App_Menu, 1032
cut
    FI_Input, 972
d
    FI_Image, 936
damage
    FI_Widget, 1883, 1884
damage_zone
    FI_Table, 1521
data
    FI_Browser, 477, 479
    FI_Image, 936
decorate
    FI_Menu_Item, 1082
    FI_Tree_Item, 1801
    FI_Widget, 1884
debug
    FI_Grid, 836
DEBUG_FLAG
    FI_Anim_GIF_Image, 435
decorated_h
    FI_Window, 1925
decorated_w
    FI_Window, 1925
default_atclose
    Windows handling functions, 267
default_callback
    FI_Widget, 1884
default_cursor
    FI_Widget, 1926
default_icon
    FI_Window, 1926
default_icons
    FI_Window, 1926, 1927
default_size_range
    FI_Window, 1927
default_xclass
    FI_Window, 1928
desaturate
    FI_Anim_GIF_Image, 438
    FI_Image, 937
    FI_Pixmap, 1196
    FI_RGB_Image, 1313
    FI_Shared_Image, 1408
    FI_TVGL Image, 1486
    FI_Tiled_Image, 1716
deselect
    FI_Browser, 506
    FI_Tree, 1767, 1768
deselect_all
    FI_Tree, 1768
    FI_Tree_Item, 1801
Designing a Simple Text Editor, 39
Developer info for bundled libs, 227
Developer Information, 231
Development of the FLTK library, 209
DIM
    FI_Terminal, 1585
DIM_CURSOR
    FI_Text_Display, 1639
dingbats_.h, 2316
direction
    FI_Timer, 1731
DIRECTORY
    FI_File_Chooser, 687
directory
    FI_Native_File_Chooser, 1141
dirty
    FI_Preferences, 1237
disable
    FI_Tooltip, 1742
disable_im
    Events handling functions, 273
display
    FI, 407
INDEX

Fl_Browser, 479
Fl_Browser_, 507
Fl_Tree, 1769
display_columns
Fl_Terminal, 1591
display_insert
Fl_Text_Display, 1644
display_rows
Fl_Terminal, 1592
displayed
Fl_Browser, 479
Fl_Browser_, 507
Fl_Tree, 1769
dnd
Selection & Clipboard functions, 284
dnd_text_ops
Fl, 407
do_callback
Fl_Menu_Item, 1082
Fl_Table, 1521
Fl_Widget, 1886
do_widget_deletion
Safe widget deletion support functions, 341
DONT_RESIZE_CANVAS
Fl_Anim_GIF_Image, 435
DONT_SET_AS_IMAGE
Fl_Anim_GIF_Image, 435
DONT_START
Fl_Anim_GIF_Image, 435
down_box
Fl_Button, 527
Fl_File_Input, 705
Fl_Menu_, 1045
drag_intersection
Fl_Tile, 1707
draw
Fl_Adjuster, 428
Fl_Anim_GIF_Image, 438
Fl_Bitmap, 449
Fl_Box, 461
Fl_Browser_, 507
Fl_Button, 527
Fl_Cairo_Window, 543
Fl_Chart, 553
Fl_Choice, 590
Fl_Clock_Output, 609
Fl_Counter, 637
Fl_Dial, 648
Fl_File_Icon, 692
Fl_File_Input, 705
Fl_Flex, 735
Fl_FormsBitmap, 757
Fl_FormsPixmap, 764
Fl_FormsText, 771
Fl_Free, 778
Fl_Gl_Window, 800
Fl_Glut_Window, 821
Fl_Grid, 837
Fl_Group, 855
Fl_Help_View, 878
Fl_Image, 937
Fl_Input, 959
Fl_Label, 1014
Fl_Light_Button, 1022
Fl_Menu_Bar, 1063
Fl_Menu_Button, 1075
Fl_Pack, 1185
Fl_Pixmap, 1196
Fl_Positioner, 1218
Fl_Progress, 1264
Fl_Return_Button, 1307
Fl_RGB_Image, 1313
Fl_Roller, 1323
Fl_Scroll, 1362
Fl_Scrollbar, 1375
Fl_Shared_Image, 1409
Fl_Shortcut_Button, 1420
Fl_Slider, 1450
Fl_Spinner, 1461
Fl_SVG_Image, 1487
Fl_Sys_Menu_Bar, 1500
Fl_Table, 1521
Fl_Tabs, 1558
Fl_Terminal, 1592
Fl_Text_Display, 1644
Fl_Tiled_Image, 1716
Fl_Timer, 1731
Fl_Tree, 1769
Fl_Tree_Item, 1801
Fl_Value_Input, 1842
Fl_Value_Output, 1853
Fl_Value_Slider, 1864
Fl_Widget, 1887
Fl_Widget_Surface, 1907
Fl_Window, 1929
Fl_Wizard, 1949
draw_bar_chart
Fl_Chart, 553
draw_begin
Fl_Gl_Window, 801
draw_box activates
Fl, 407
draw_buffer
Fl_Terminal, 1592
draw_cell
Fl_Table, 1521
draw_child
Fl_Group, 856
draw_children
Fl_Group, 856
draw_cursor
Fl_Text_Display, 1644
draw_decorated_window
Fl_Widget_Surface, 1907
draw_empty
Fl_Image, 937

Generated by Doxygen
draw_end
  Fl_Gl_Window, 801

draw_focus
  Fl_Widget, 1887, 1888

draw_GL_text_with_textures
  Fl, 408

draw_grid
  Fl_Grid, 837

draw_horbarchart
  Fl_Chart, 554

draw_horizontal_connector
  Fl_Tree_Item, 1802

draw_item_content
  Fl_Tree_Item, 1802

draw_label
  Fl_Widget, 1888, 1889

draw_line_numbers
  Fl_Text_Display, 1644

draw_linechart
  Fl_Chart, 555

draw_overlay
  Fl_Glut_Window, 821
  Fl_Overlay_Window, 1175

draw_piechart
  Fl_Chart, 555

draw_range
  Fl_Text_Display, 1645

draw_row
  Fl_Terminal, 1592

draw_row_bg
  Fl_Terminal, 1592

draw_scaled
  Fl_Image, 937

draw_string
  Fl_Text_Display, 1645

draw_tab
  Fl_Tabs, 1558

draw_text
  Fl_Text_Display, 1645

draw_vertical_connector
  Fl_Tree_Item, 1803

draw_vline
  Fl_Text_Display, 1646

drawbgcolor
  Fl_Tree_Item, 1803

drawfgcolor
  Fl_Tree_Item, 1804

Drawing functions, 306
  fl_add_symbol, 311
  fl_antialias, 312
  fl_arc, 312, 313
  fl_begin_complex_polygon, 313
  fl_begin_offscreen, 313
  fl_begin_points, 314
  fl_can_do_alpha_blending, 314
  FL_CAP_FLAT, 311
  FL_CAP_ROUND, 311
  FL_CAP_SQUARE, 311
  fl_capture_window, 314
  fl_circle, 314
  fl_clip, 315
  fl_clip_box, 315
  fl_clip_region, 316
  fl_copy_offscreen, 316
  fl_create_offscreen, 317
  fl_cursor, 317
  fl_curve, 317
  FL_DASH, 311
  FL_DASHDOT, 311
  FL_DASHDOTDOT, 311
  fl_delete_offscreen, 318
  FL_DOT, 311
  fl_draw, 318, 319
  fl_draw_arrow, 319
  fl_draw_box, 319
  fl_draw_check, 320
  fl_draw_circle, 320
  fl_draw_image, 321
  fl_draw_image_mono, 322
  fl_draw_pixmap, 322, 323
  fl_draw_radio, 323
  fl_draw_symbol, 323
  fl_expand_text, 324
  fl_focus_rect, 324
  fl_frame, 324
  fl_frame2, 325
  fl_gap, 325
  FL_JOIN_BEVEL, 311
  FL_JOIN_MITER, 311
  FL_JOIN_ROUND, 311
  fl_line_style, 325
  fl_load_matrix, 326
  fl_measure, 326
  fl_measure_pixmap, 326, 327
  fl_mult_matrix, 327
  fl_not_clipped, 327
  fl_old_shortcut, 328
  fl_overview_clear, 328
  fl_overview_rect, 329
  fl_override_scale, 329
  fl_pie, 330
  fl_polygon, 330
  fl_pop_clip, 331
  fl_push_clip, 331
  fl_push_matrix, 331
  fl_read_image, 331
  fl_rect, 332
  fl_rectf, 332, 333
  fl_rescale_offscreen, 333
  fl_reset_spot, 333
  fl_restore_scale, 333
  fl_rotate, 334
  fl_rounded_rect, 334
  fl_rounded_rectf, 334
  fl_scale, 334
  fl_scroll, 335
INDEX

fl_set_spot, 335
fl_set_status, 335
fl_shortcut_label, 335, 336
FL_SOLID, 311
fl_transform_dx, 337
fl_transform_dy, 337
fl_transform_x, 337
fl_transform_y, 337
fl_transformed_vertex, 337
fl_translate, 338
fl_vertex, 338

Drawing Things in FLTK, 53
drawtext
   FL_Input, 972, 973
dvalue
   FL_Input, 973
elapse_timeouts
   FL_Timeout, 1720
empty_vlines
   FL_Text_Display, 1646
enable
   FL_Tooltip, 1742
enable_im
   Events handling functions, 273
enabled
   FL_Tooltip, 1742
end
   FL_Flex, 735
   FL_Group, 856
   FL_Text_Selection, 1693
end_current
   FL_PostScript_File_Device, 1224
   FL_Surface_Device, 1475
end_job
   FL_Paged_Device, 1190
   FL_PostScript_File_Device, 1224
   FL_Printer, 1254
end_page
   FL_Paged_Device, 1191
   FL_PostScript_File_Device, 1225
   FL_Printer, 1254
enter_area
   FL_Tooltip, 1742
entries
   FL_Preferences, 1237
entry
   FL_Preferences, 1237
entry_exists
   FL_Preferences, 1238
   _FL_DIAMOND_DOWN_BOX, 1980
   _FL_DIAMOND_UP_BOX, 1980
   _FL_EMBOSSED_LABEL, 1986
   _FL_ENGRAVED_LABEL, 1986
   _FL_GLEAM_DOWN_BOX, 1980
   _FL_GLEAM_DOWN_FRAME, 1981
   _FL_GLEAM_ROUND_DOWN_BOX, 1981
   _FL_GLEAM_ROUND_DOWN_FRAME, 1981
   _FL_GLEAM_THIN_DOWN_BOX, 1981
   _FL_GLEAM_THIN_UP_BOX, 1981
   _FL_GTK_DOWN_BOX, 1980
   _FL_GTK_DOWN_FRAME, 1980
   _FL_GTK_ROUND_DOWN_BOX, 1980
   _FL_GTK_ROUND_UP_BOX, 1980
   _FL_GTK_THIN_DOWN_BOX, 1980
   _FL_GTK_THIN_DOWN_FRAME, 1980
   _FL_GTK_THIN_UP_BOX, 1980
   _FL_GTK_THIN_UP_FRAME, 1980
   _FL_GTK_UP_BOX, 1980
   _FL_GTK_UP_FRAME, 1980
   _FL_ICON_LABEL, 1986
   _FL_IMAGE_LABEL, 1986
   _FL_MULTI_LABEL, 1986
   _FL_OFLAT_BOX, 1980
   _FL_OSHADOW_BOX, 1980
   _FL_OVAL_BOX, 1980
   _FL_OVAL_FRAME, 1980
   _FL_OXY_BUTTON_DOWN_BOX, 1981
   _FL_OXY_BUTTON_UP_BOX, 1981
   _FL_OXY_DOWN_BOX, 1981
   _FL_OXY_DOWN_FRAME, 1981
   _FL_OXY_ROUND_DOWN_BOX, 1981
   _FL_OXY_ROUND_UP_BOX, 1981
   _FL_OXY_THIN_DOWN_BOX, 1981
   _FL_OXY_THIN_DOWN_FRAME, 1981
   _FL_OXY_THIN_UP_BOX, 1981
   _FL_OXY_THIN_UP_FRAME, 1981
   _FL_OXY_UP_BOX, 1981
   _FL_OXY_UP_FRAME, 1981
   _FL_PLASTIC_DOWN_BOX, 1980
   _FL_PLASTIC_DOWN_FRAME, 1980
   _FL_PLASTIC_ROUND_DOWN_BOX, 1980
   _FL_PLASTIC_ROUND_UP_BOX, 1980
   _FL_PLASTIC_THIN_DOWN_BOX, 1980
   _FL_PLASTIC_THIN_UP_BOX, 1980
   _FL_PLASTIC_UP_BOX, 1980
   _FL_PLASTIC_UP_FRAME, 1980
   _FL_RFLAT_BOX, 1980
   _FL_ROUNDED_BOX, 1980
   _FL_ROUNDED_FRAME, 1980
   _FL_ROUND_DOWN_BOX, 1980
   _FL_ROUND_UP_BOX, 1980
   _FL_RSHADOW_BOX, 1980
   _FL_SHADOW_BOX, 1980
   _FL_SHADOW_FRAME, 1980
   _FL_SHADOW_LABEL, 1986
FL_ABI_VERSION, 1976
FL_ACTIVATE, 1985
FL_ALIGN_LEFT, 1989
FL_ALIGN_TOP, 1989
FL_API_VERSION, 1977
FL_ARROW_CHOICE, 1979
FL_ARROW_DOUBLE, 1979
FL_ARROW_RETURN, 1979

Generated by Doxygen
FL_ARROW_SINGLE, 1979
Fl_Arrow_Type, 1979
FL_BORDER_BOX, 1980
FL_BORDER_FRAME, 1980
fl_box, 1988
Fl_Boxtype, 1979
Fl_Callback_Reason, 1981
FL_CLOSE, 1984
fl_color_cube, 1988
FL_CONTRAST_CIELAB, 1982
FL_CONTRAST_CUSTOM, 1982
Fl_Contrast_Function, 1978
FL_CONTRAST_LAST, 1982
FL_CONTRAST_LEGACY, 1982
Fl_Contrast_Mode, 1982
FL_CONTRAST_NONE, 1982
Fl_Cursor, 1982
FL_CURSOR_ARROW, 1982
FL_CURSOR_CROSS, 1982
FL_CURSOR_DEFAULT, 1982
FL_CURSOR_E, 1982
FL_CURSOR_HAND, 1982
FL_CURSOR_HELP, 1982
FL_CURSOR_INSERT, 1982
FL_CURSOR_MOVE, 1982
FL_CURSOR_N, 1982
FL_CURSOR_NE, 1982
FL_CURSOR_NESW, 1982
FL_CURSOR_NONE, 1982
FL_CURSOR_NS, 1982
FL_CURSOR_NW, 1982
FL_CURSOR_NWSE, 1982
FL_CURSOR_S, 1982
FL_CURSOR_SE, 1982
FL_CURSOR_SW, 1982
FL_CURSOR_W, 1982
FL_CURSOR_WAIT, 1982
FL_CURSOR_WE, 1982
Fl_Damage, 1983
FL_DAMAGE_ALL, 1983
FL_DAMAGE_CHILD, 1983
FL_DAMAGE_EXPOSE, 1983
FL_DAMAGE_OVERLAY, 1983
FL_DAMAGE_SCROLL, 1983
FL_DAMAGE_USER1, 1983
FL_DAMAGE_USER2, 1983
FL_DEACTIVATE, 1985
fl_define_FL_EMBOSSED_LABEL, 1988
fl_define_FL_ENGRAVED_LABEL, 1988
fl_define_FL_ICON_LABEL, 1988
fl_define_FL_IMAGE_LABEL, 1988
fl_define_FL_MULTI_LABEL, 1988
fl_define_FL_SHADOW_LABEL, 1988
FL_DND_DRAG, 1985
FL_DND_ENTER, 1985
FL_DND_LEAVE, 1985
FL_DND_RELEASE, 1985
fl_down, 1988
FL_DOWN_BOX, 1979
FL_DOWN_FRAME, 1979
FL_DRAG, 1984
FL_EMBOSSED_BOX, 1980
FL_EMBOSSED_FRAME, 1980
FL_ENGRAVED_BOX, 1979
FL_ENGRAVED_FRAME, 1980
FL_EVENT, 1983
Fl_Event, 1983
FL_EXCEPTION, 1979
FL_FLAT_BOX, 1979
FL_FOCUS, 1984
Fl_Fontsize, 1978
fl_frame, 1989
FL_FREE_BOXTYPE, 1981
FL_FREE_LABELTYPE, 1986
FL_FULLSCREEN, 1986
fl_gray_ramp, 1989
FL_HIDE, 1985
FL_IMAGE_LABEL, 1977
FL_KEYBOARD, 1984
FL_KEYDOWN, 1984
FL_KEYUP, 1984
Fl_Labeltype, 1986
FL_LEAVE, 1984
FL_MAJOR_VERSION, 1977
FL_MINOR_VERSION, 1977
FL_MOUSEWHEEL, 1985
FL_MOVE, 1985
FL_MULTI_LABEL, 1977
FL_NO_BOX, 1979
FL_NO_EVENT, 1983
FL_NO_LABEL, 1986
FL_NORMAL_LABEL, 1986
FL_NORMAL_SIZE, 1989
FL_ORIENTATION, 1987
FL_ORIENTATION_LEFT, 1987
FL_ORIENTATION_NE, 1987
FL_ORIENTATION_NONE, 1987
FL_ORIENTATION_NW, 1987
FL_ORIENTATION_RIGHT, 1987
FL_ORIENTATION_SE, 1987
FL_ORIENTATION_SW, 1987
FL_ORIENTATION_UP, 1987
Fl_Orientation, 1986
FL_PASTE, 1985
FL_PATCH_VERSION, 1977
FL_PUSH, 1983
FL_READ, 1979
FL_REASON_CANCELLED, 1981
FL_REASON_CHANGED, 1982
FL_REASON_CLOSED, 1981
FL_REASON_DESELECTED, 1981
FL_REASON_DRAGGED, 1981
FL_REASON_ENTER_KEY, 1982
FL_REASON_GOT_FOCUS, 1982
FL_REASON_LOST_FOCUS, 1982
FL_REASON_OPENED, 1981
INDEX 2665

FL_REASON_RELEASED, 1982
FL_REASON_RESELECTED, 1981
FL_REASON_SELECTED, 1981
FL_REASON_USER, 1982
FL_RELEASE, 1983
FL_SCREEN_CONFIGURATION_CHANGED, 1986
FL_SELECTIONCLEAR, 1985
FL_SHORTCUT, 1985
FL_SHOW, 1985
FL_SYMBOL_LABEL, 1977
FL_THIN_DOWN_BOX, 1979
FL_THIN_DOWN_FRAME, 1979
FL_THIN_UP_BOX, 1979
FL_THIN_UP_FRAME, 1979
FL_UNFOCUS, 1984
FL_UP_BOX, 1979
FL_UP_FRAME, 1979
FL_VERSION, 1978
FL_When, 1987
FL_WHEN_CHANGED, 1987
FL_WHEN_CLOSED, 1987
FL_WHEN_ENTER_KEY, 1987
FL_WHEN_ENTER_KEY_ALWAYS, 1987
FL_WHEN_ENTER_KEY_CHANGED, 1987
FL_WHEN_NEVER, 1987
FL_WHEN_NOT_CHANGED, 1987
FL_WHEN_RELEASE, 1987
FL_WHEN_RELEASE_ALWAYS, 1987
FL_WRITE, 1979
FL_ZOOM_EVENT, 1986
FL_ZOOM_GESTURE, 1986
EOL
FI_Terminal, 1586
errmsg
Fl_File_Browser, 681
Fl_Native_File_Chooser, 1141
error
Common Dialog Classes and Functions, 377
errorcolor
Fl_File_Input, 705
ERRORS_TO_CP1252
Unicode and UTF-8 functions, 347
ERRORS_TO_ISO8859_1
Unicode and UTF-8 functions, 347
event
Events handling functions, 273
event_button
Events handling functions, 273
event_button1
Events handling functions, 273
event_button2
Events handling functions, 274
event_button3
Events handling functions, 274
event_buttons
Events handling functions, 274
event_clicks
Events handling functions, 274
event_clipboard
Events handling functions, 274
event_clipboard_type
Events handling functions, 274
event_dispatch
Events handling functions, 275
event_dx
Events handling functions, 275
event_dy
Events handling functions, 275
event_inside
Events handling functions, 275, 276
event_is_click
Events handling functions, 276
event_key
Events handling functions, 276, 277
event_length
Events handling functions, 277
event_original_key
Events handling functions, 277
event_state
Events handling functions, 277, 278
event_text
Events handling functions, 278
event_x_root
Events handling functions, 278
event_y_root
Events handling functions, 278
Events handling functions, 268
add_handler, 271
add_system_handler, 271
belowmouse, 271, 272
callback_reason, 272
compose, 272
compose_reset, 273
disable_im, 273
able_im, 273
event, 273
event_button, 273
event_button1, 273
event_button2, 274
event_button3, 274
event_buttons, 274
event_clicks, 274
event_clipboard, 274
event_clipboard_type, 274
event_dispatch, 275
event_dx, 275
event_dy, 275
event_inside, 275, 276
event_is_click, 276
event_key, 276, 277
event_length, 277
event_original_key, 277
event_state, 277, 278
event_text, 278

Generated by Doxygen
INDEX 2667

fixed
   FL_Flex, 735, 736
FL, 390
   abi_check, 400
   abi_version, 400
   add_check, 400
   add_fd, 401
   add_idle, 401
   add_timeout, 401
   api_version, 402
   arg, 402
   args, 403
   args_to_utf8, 404
   background, 405
   background2, 405
   box_border_radius_max, 405
   box_color, 406
   box_dh, 406
   box_dw, 406
   box_dx, 406
   box_dy, 406
   box_shadow_width, 407
   check, 407
   display, 407
   dnd_text_ops, 407
   draw_box_active, 407
   draw_GL_text_with_textures, 408
   FL_Option, 399
   flush, 408
   get_system_colors, 408
   gl_visual, 409
   has_timeout, 409
   help, 419
   hide_all_windows, 409
   idle, 419
   is_scheme, 410
   menu_linespacing, 410
   now, 410
   option, 411, 412
   OPTION_ARROW_FOCUS, 399
   OPTION_DND_TEXT, 399
   OPTION_FNFC_USES_GTK, 400
   OPTION_FNFC_USES_ZENITY, 400
   OPTION_LAST, 400
   OPTION_PRINTER_USES_GTK, 400
   OPTION_SHOW_SCALING, 400
   OPTION_SHOW_TOOLTIPS, 399
   OPTION_VISIBLE_FOCUS, 399
   own_colormap, 412
   program_should_quit, 412
   readqueue, 413
   ready, 413
   release, 413
   reload_scheme, 413
   remove_check, 413
   remove_timeout, 414
   repeat_timeout, 414
   run, 414
   scheme, 415
   scrollbar_size, 415, 416
   seconds_between, 416
   seconds_since, 416
   set_box_color, 416
   set_idle, 417
   ticks_between, 417
   ticks_since, 417
   use_high_res_GL, 418
   version, 418
   visible_focus, 418
   visual, 418
   wait, 419
FL.cxx, 2248
   fl_disable_wayland, 2250
   fl_find, 2249
   fl_open_display, 2249
FL.H, 2000, 2002
FL_ABI_VERSION
   Enumerations.H, 1976
fl_access
   Unicode and UTF-8 functions, 347
FL_ACTIVATE
   Enumerations.H, 1985
fl_add_symbol
   Drawing functions, 311
FL_Adjuster, 420
   draw, 428
   Fl_Adjuster, 428
   handle, 428
   soft, 429
   value_damage, 429
FL_Adjuster.H, 2008
fl_alert
   Common Dialog Classes and Functions, 364
FL_ALIGN_LEFT
   Enumerations.H, 1989
FL_ALIGN_TOP
   Enumerations.H, 1989
FL_Anim_GIF_Image, 429
   ~FL_Anim_GIF_Image, 436
   canvas, 436, 437
   canvas_h, 437
   canvas_w, 437
   color_average, 437
   copy, 437
   DEBUG_FLAG, 435
   delay, 438
   desaturate, 438
   DONT_RESIZE_CANVAS, 435
   DONT_SET_AS_IMAGE, 435
   DONT_START, 435
   draw, 438
   Fl_Anim_GIF_Image, 435, 436
   Flags, 435
   frame, 439
   frame_count, 439
   frame_h, 440

Generated by Doxygen
_FL_METHOD_CALLBACK_1, 2027
Fl_Callback_Reason
Enumerations.H, 1981
fl_callback_reason_names
Events handling functions, 281
Fl_Callback_User_Data, 543
fl_can_do_alpha_blending
Drawing functions, 314
FL_CAP_FLAT
Drawing functions, 311
FL_CAP_ROUND
Drawing functions, 311
FL_CAP_SQUARE
Drawing functions, 311
fl_capture_window
Drawing functions, 314
fl_casenumericsort
numericsort.c, 2290
fl_casts.H, 2034
Fl_Chart, 544
add, 552
autosize, 553
bounds, 553
draw, 553
draw_barchart, 553
draw_horbarchart, 554
draw_linechart, 555
draw_piechart, 555
Fl_Chart, 552
insert, 555
maxsize, 556
replace, 556
size, 556
Fl_Chart.H, 2034, 2035
FL_CHART_ENTRY, 557
fl_chdir
Unicode and UTF-8 functions, 348
Fl_Check_Browser, 557
add, 569
handle, 569
item_at, 569
item_draw, 569
item_first, 570
item_height, 570
item_next, 570
item_prev, 570
item_select, 570
item_selected, 571
item_swap, 571
item_text, 571
item_width, 572
nitems, 572
remove, 572
Fl_Check_Browser.H, 2036
Fl_Check_Button, 572
Fl_Check_Button.H, 2037
fl_chmod
Unicode and UTF-8 functions, 348
Fl_Choice, 580
draw, 590
Fl_Choice, 590
handle, 591
value, 591, 592
fl_choice
Common Dialog Classes and Functions, 365
Fl_Choice.H, 2038
fl_choice_n
Common Dialog Classes and Functions, 367
fl_circle
Drawing functions, 314
fl_clip
Drawing functions, 315
fl_clip_box
Drawing functions, 315
fl_clip_region
Drawing functions, 316
Fl_Clock, 592
Fl_Clock, 600
handle, 600
Fl_Clock.H, 2038
Fl_Clock_Output, 601
draw, 609
Fl_Clock_Output, 608
hour, 609
minute, 609
second, 609
shadow, 610
value, 610, 611
FL_CLOSE
Enumerations.H, 1984
fl_close_fd
Unicode and UTF-8 functions, 349
fl_cmap
fl_color.cxx, 2257
fl_cmap.h, 2254
fl_color
Color & Font functions, 294
fl_color.cxx, 2257
fl_cmap, 2257
fl_color_average
Color & Font functions, 295
Fl_Color_Chooser, 611
b, 621
Fl_Color_Chooser, 620
g, 621
handle, 621
hsv, 621
hsv2rgb, 622
hue, 622
mode, 622
r, 622
rgb, 622
rgb2hsv, 623
saturation, 623
value, 623
INDEX 2671

fl_color_chooser  
Common Dialog Classes and Functions, 367, 368
Fl_Color_Chooser.H, 2039, 2040

fl_color_cube  
Enumerations.H, 1988
Fl-compose.cxx, 2257
fl_config.h, 2041

fl_contrast  
Color & Font functions, 295
Fl_contrast.cxx, 2258
FL_CONTRAST_CIELAB  
Enumerations.H, 1982
FL_CONTRAST_CUSTOM  
Enumerations.H, 1982
Fl_Contrast_Function  
Enumerations.H, 1978
fl_contrast_function  
Color & Font functions, 296
FL_CONTRAST_LAST  
Enumerations.H, 1982
FL_CONTRAST_LEGACY  
Enumerations.H, 1982
fl_contrast_level  
Color & Font functions, 297
Fl_Contrast_Mode  
Enumerations.H, 1982
fl_contrast_mode  
Color & Font functions, 298
FL_CONTRAST_NONE  
Enumerations.H, 1982
fl_copy_offscreen  
Drawing functions, 316
Fl_Copy_Surface, 623
Fl_Copy_Surface, 626
is_current, 626
origin, 626
printable_rect, 626
set_current, 627
translate, 627
untranslate, 627
Fl_Copy_Surface.H, 2042
Fl_Counter, 627
arrow_widths, 636
draw, 637
Fl_Counter, 636
handle, 637
lstep, 637
step, 638
Fl_Counter.H, 2043
fl_create_offscreen  
Drawing functions, 317
Fl_Cursor  
Enumerations.H, 1982
fl_cursor  
Drawing functions, 317
FL_CURSOR_ARROW  
Enumerations.H, 1982
FL_CURSOR_DEFAULT  
Enumerations.H, 1982
FL_CURSOR_E  
Enumerations.H, 1982
FL_CURSOR_HAND  
Enumerations.H, 1982
FL_CURSOR_HELP  
Enumerations.H, 1982
FL_CURSOR_INSERT  
Enumerations.H, 1982
FL_CURSOR_MOVE  
Enumerations.H, 1982
FL_CURSOR_N  
Enumerations.H, 1982
FL_CURSOR_NE  
Enumerations.H, 1982
FL_CURSOR_NESW  
Enumerations.H, 1982
FL_CURSOR_NONE  
Enumerations.H, 1982
FL_CURSOR_NW  
Enumerations.H, 1982
FL_CURSOR_NWSE  
Enumerations.H, 1982
FL_CURSOR_S  
Enumerations.H, 1982
FL_CURSOR_SE  
Enumerations.H, 1982
FL_CURSOR_SW  
Enumerations.H, 1982
FL_CURSOR_W  
Enumerations.H, 1982
FL_CURSOR_WAIT  
Enumerations.H, 1982
FL_CURSOR_WE  
Enumerations.H, 1982
FL_CURSOR_WE  
Enumerations.H, 1982
fl_curve  
Drawing functions, 317
fl_curve.cxx, 2258
Fl_Damage  
Enumerations.H, 1983
FL_DAMAGE_ALL  
Enumerations.H, 1983
FL_DAMAGE_CHILD  
Enumerations.H, 1983
FL_DAMAGE_EXPOSE  
Enumerations.H, 1983
FL_DAMAGE_OVERLAY  
Enumerations.H, 1983
FL_DAMAGE_SCROLL  
Enumerations.H, 1983
FL_DAMAGE_USER1  
Enumerations.H, 1983
FL_DAMAGE_USER2  
Enumerations.H, 1983

Generated by Doxygen
FL_DASH
Drawing functions, 311

FL_DASHDOT
Drawing functions, 311

FL_DASHDOTDOT
Drawing functions, 311

FL_DEACTIVATE
Enumerations.H, 1985

fl_decode_uri
File names and URI utility functions, 379

fl_define_FL_EMBOSSED_LABEL
Enumerations.H, 1988

fl_define_FL_ENGRAVED_LABEL
Enumerations.H, 1988

fl_define_FL_ICON_LABEL
Enumerations.H, 1988

fl_define_FL_IMAGE_LABEL
Enumerations.H, 1988

fl_define_FL_MULTI_LABEL
Enumerations.H, 1988

fl_define_FL_SHADOW_LABEL
Enumerations.H, 1988

fl_delete_offscreen
Drawing functions, 318

FL_DEPRECATED
fl_attr.h, 2014

Fl_Device.H, 2044

Fl_Device_Pi58, 638
rectangle_capture, 639

Fl_Dial, 639
angle1, 648
draw, 648
Fl_Dial, 647
handle, 648

Fl_Dial.H, 2045

fl_dir_chooser
Common Dialog Classes and Functions, 369

fl_disable_wayland
Fl.cxx, 2250

FL_Double_Window, 648

FL_Double_Window.H, 2046

fl_down
Enumerations.H, 1988

FL_DOWN_BOX
Enumerations.H, 1979

FL_DOWN_FRAME
Enumerations.H, 1979

FL_DRA3
Enumerations.H, 1984

fl_draw
Drawing functions, 318, 319

fl_draw.H, 2046, 2052

fl_draw_arrow
Drawing functions, 319

fl_draw_box
Drawing functions, 319

fl_draw_check
Drawing functions, 320

fl_draw_circle
Drawing functions, 320

fl_draw_image
Drawing functions, 321

fl_draw_image_mono
Drawing functions, 322

fl_draw_pixmap
Drawing functions, 322, 323

fl_draw_radio
Drawing functions, 323

fl_draw_symbol
Drawing functions, 323

FL_EMBOSSED_BOX
Enumerations.H, 1980

FL_EMBOSSED_FRAME
Enumerations.H, 1980

FI_End, 663

FL_ENGRAVED_BOX
Enumerations.H, 1979

FL_ENGRAVED_FRAME
Enumerations.H, 1980

FL_ENTER
Enumerations.H, 1983

Fl_EPS_File_Surface, 664
~Fl_EPS_File_Surface, 666
close, 666
Fl_EPS_File_Surface, 666
origin, 666
printable_rect, 667
translate, 667
untranslate, 667

Fl_Event
Enumerations.H, 1983

Fl_Event_Dispatch
Callback Function Typedefs, 266

fl_eventnames
Events handling functions, 281

FL_EXCEPT
Enumerations.H, 1979

fl_expand_text
INDEX 2673

Drawing functions, 324
FL_Export.H, 2058
FL_File_Browser, 667
   errmsg, 681
   filetype, 681
   filter, 681
   FL_File_Browser, 681
   iconsize, 682
   load, 682
FL_File_Browser.H, 2058
FL_File_Chooser, 682
   add_extra, 688
   CREATE, 687
   DIRECTORY, 687
   filter, 688
   FL_File_Chooser, 687
   iconsize, 688, 689
   MULTI, 687
   preview, 689
   showHiddenButton, 689
   shown, 689
   SINGLE, 687
   Type, 687
   value, 689
fl_file_chooser
   Common Dialog Classes and Functions, 370
FL_File_Chooser.H, 2059
fl_file_chooser_callback
   Common Dialog Classes and Functions, 371
fl_file_chooser_ok_label
   Common Dialog Classes and Functions, 371
FL_File_Icon, 689
   add, 691
   add_color, 691
   add_vertex, 691, 692
   draw, 692
   find, 692
   FL_File_Icon, 691
   label, 692
   labeltype, 693
   load, 693
   load_fti, 693
   load_image, 693
   load_system_icons, 694
   next, 694
   type, 694
FL_File_Icon.H, 2061
FL_File_Input, 694
   down_box, 705
   draw, 705
   errorcolor, 705
   FL_File_Input, 704
   handle, 705
   value, 705, 706
FL_File_Input.H, 2063
FL_File_Sort_F
   File names and URI utility functions, 379
fl_filename_absolute
   File names and URI utility functions, 379, 380
fl_filename_expand
   File names and URI utility functions, 380
fl_filename_ext
   File names and URI utility functions, 381
fl_filename_free_list
   File names and URI utility functions, 381
fl_filename_isdir
   File names and URI utility functions, 381
fl_filename_list
   File names and URI utility functions, 381
fl_filename_match
   File names and URI utility functions, 382
fl_filename_name
   File names and URI utility functions, 383
fl_filename_relative
   File names and URI utility functions, 383, 384
fl_filename_setext
   File names and URI utility functions, 384
FL_Fill_Dial, 706
FL_Fill_Dial.H, 2063
FL_Fill_Slider, 714
FL_Fill_Slider.H, 2064
fl_find
   F1.cxx, 2249
FL_FLAT_BOX
  Enumerations.H, 1979
FL_Flex, 722
   alloc_size, 735
   COLUMN, 733
draw, 735
   end, 735
   fixed, 735, 736
   FL_Flex, 733, 734
gap, 736
   HORIZONTAL, 733
   horizontal, 737
   layout, 737
   margin, 737, 738
   need_layout, 739
   on_remove, 739
   resize, 739
   ROW, 733
   spacing, 740
   VERTICAL, 733
FL_Flex.H, 2064
FL_Float_Input, 740
   FL_Float_Input.H, 2066
FL_FOCUS
   Enumerations.H, 1984
fl_focus_rect
   Drawing functions, 324
fl_font
   Color & Font functions, 299
fl_fontnames
   Events handling functions, 282
FL_Fontsize

Generated by Doxygen
FL_INLINE_CALLBACK_2
   fl_callback_macros.H, 2026
FL_Input, 948
draw, 959
   FL_Input, 959
handle, 959
handle_key, 959
handle_rmb, 960
fl_input
   Common Dialog Classes and Functions, 371
FL_Input.H, 2091
FL_Input_, 960
~FL_Input_, 970
append, 970
apply_undo, 970
can_redo, 970
can_undo, 970

copy, 971

copy_cuts, 971
cursor_color, 971
cut, 972
drawtext, 972, 973
dvalue, 973
FL_Input_, 969
handle_mouse, 973
handletext, 973
index, 974
input_type, 974
insert, 974
insert_position, 975
ivalue, 976
line_end, 976
line_start, 976
mark, 976, 977
maximum_size, 977
position, 977
readonly, 978
redo, 978
replace, 978
resize, 979
shortcut, 979
size, 980
static_value, 980

tab_nav, 981
textcolor, 981, 982
textfont, 982
textsize, 982
undo, 984
up_down_position, 984
value, 984, 985
word_end, 986
word_start, 986
wrap, 986
FL_Input_H, 2092
FL_Input_Choice, 987
add, 997
   FL_Input_Choice, 997
inp_x, 997
input, 998
menu_x, 998
menubutton, 998
resize, 998
update_menubutton, 998
value, 999
FL_Input_Choice.H, 2095
FL_Int_Input, 999
   FL_Int_Input, 1009
FL_Int_Input.H, 2097
FL_Int_Vector.H, 2264
fl_internal_boxtype
   fl_boxtype.cxx, 2253
fl_intptr_t
   platform_types.h, 2238
FL_JOIN_BEVEL
   Drawing functions, 311
FL_JOIN_MITER
   Drawing functions, 311
FL_JOIN_ROUND
   Drawing functions, 311
FL_JPEG_Image, 1009
   FL_JPEG_Image, 1012
FL_JPEG_Image.H, 2097
FL_KEYBOARD
   Enumerations.H, 1984
FL_KEYDOWN
   Enumerations.H, 1984
FL_KEYUP
   Enumerations.H, 1984
FL_Label, 1013
draw, 1014
measure, 1014
type, 1014
FL_Labeltype
   Enumerations.H, 1986
fl_latin1_to_local
   Color & Font functions, 299
FL_LEAVE
   Enumerations.H, 1984
FL_Light_Button, 1014
draw, 1022
   FL_Light_Button, 1022
handle, 1022
FL_Light_Button.H, 2097
fl_lightness
   Color & Font functions, 300
FL_Line_Dial, 1023
FL_Line_Dial.H, 2098
fl_line_style
   Drawing functions, 325
fl_load_matrix
   Drawing functions, 326
fl_local_to_latin1
   Color & Font functions, 300
fl_local_to_mac_roman
   Color & Font functions, 300
fl_luminance
INDEX 2677

Color & Font functions, 301
Fl_Mac_App_Menu, 1031
custom_application_menu_items, 1032
print, 1032
fl_mac_os_version
Mac OS X-specific symbols, 362
fl_mac_roman_to_local
Color & Font functions, 301
fl_mac_set_about
Mac OS X-specific symbols, 362
FL_MAJOR_VERSION
Enumerations.H, 1977
fl_make_path
Unicode and UTF-8 functions, 350
fl_make_path_for_file
Unicode and UTF-8 functions, 350
fl_measure
Drawing functions, 326
fl_measure_pixmap
Drawing functions, 326, 327
Fl_Menu.H, 2098
Fl_Menu_.H, 1032
add, 1041, 1042
clear, 1044
clear_submenu, 1044
copy, 1045
down_box, 1045
find_index, 1045, 1046
find_item, 1047
find_item_with_argument, 1047
find_item_with_user_data, 1048
Fl_Menu_.H, 1041
global, 1048
insert, 1048
item_pathname, 1049
menu, 1049, 1050
menu_box, 1050
menu_end, 1050
mode, 1051
mvalue, 1051
picked, 1051
prev_mvalue, 1051
remove, 1051
replace, 1052
size, 1052
test_shortcut, 1052
text, 1052
textcolor, 1053
textfont, 1053
textsize, 1053
value, 1053
Fl_Menu_.H, 2099
Fl_Menu_Bar, 1054
draw, 1063
Fl_Menu_Bar_H, 2100
Fl_Menu_Bar, 1063
draw, 1063
handle, 1064
update, 1064
Fl_Menu_Bar.H, 2100
Fl_Menu_Button, 1064
draw, 1075
handle, 1075
popup, 1075
POPUP1, 1074
POPUP12, 1074
POPUP123, 1074
POPUP13, 1074
POPUP2, 1074
POPUP23, 1074
POPUP3, 1074
popup_buttons, 1074
Fl_Menu_Button.H, 2100
FL_MENU_DIVIDER
Fl_Menu_Item.H, 2102
FL_MENU_HORIZONTAL
Fl_Menu_Item.H, 2102
FL_MENU_INACTIVE
Fl_Menu_Item.H, 2102
FL_MENU_INVISIBLE
Fl_Menu_Item.H, 2102
Fl_Menu_Item, 1076
add, 1080
argument, 1081
callback, 1081
check, 1082
checkbutton, 1082
checked, 1082
decomponent, 1082
do_callback, 1082
find_shortcut, 1083
image, 1083
image_label, 1083
insert, 1083
label, 1084, 1085
labelcolor, 1085
labelfont, 1085
labeltype, 1086
measure, 1086
multi_label, 1086
next, 1086
popup, 1086
pulldown, 1087
radio, 1087
set, 1087
setonly, 1087
shortcut, 1088
size, 1088
submenu, 1088
test_shortcut, 1088
uncheck, 1088
value, 1089
Fl_Menu_Item.H, 2101, 2102
FL_MENU_DIVIDER, 2102
FL_MENU_HORIZONTAL, 2102
FL_MENU_INACTIVE, 2102
FL_MENU_INVISIBLE, 2102

Generated by Doxygen
INDEX

Unicode and UTF-8 functions, 351
fl_open_callback
  Mac OS X-specific symbols, 362
fl_open_display
  Fl.cxx, 2249
fl_open_ext
  Unicode and UTF-8 functions, 351
fl_open_uri
  File names and URI utility functions, 385
Fl_Option
  Fl, 399
FL_ORIENTATION_DOWN
  Enumerations.H, 1987
FL_ORIENTATION_LEFT
  Enumerations.H, 1987
FL_ORIENTATION_NE
  Enumerations.H, 1987
FL_ORIENTATION_NONE
  Enumerations.H, 1987
FL_ORIENTATION_NW
  Enumerations.H, 1987
FL_ORIENTATION_RIGHT
  Enumerations.H, 1987
FL_ORIENTATION_SE
  Enumerations.H, 1987
FL_ORIENTATION_SW
  Enumerations.H, 1987
FL_ORIENTATION_UP
  Enumerations.H, 1987
Fl_Orientation
  Enumerations.H, 1986
Fl_Output
  Fl_Output, 1152
    Fl_Output, 1162
fl_overlay_clear
  Drawing functions, 328
fl_overlay_rect
  Drawing functions, 329
Fl_Overlay_Window
  Fl_Overlay_Window, 1162
    as_overlay_window, 1175
draw_overlay, 1175
Fl_Overlay_Window
  Fl_Overlay_Window, 1174
    flush, 1175
hide, 1175
re redraw_overlay, 1175
resize, 1175
show, 1176
Fl_Overlay_Window
  Fl_Overlay_Window, 2110
fl_override_scale
  Drawing functions, 329
fl_oxy.h
  2268
Fl_Pack
  Fl_Pack, 1176
draw, 1185
  Fl_Pack, 1185
horizontal, 1185
resize, 1186
Fl_Pack.H
  Fl_Pack.H, 2111
Fl_Paged_Device
  Fl_Paged_Device, 1186

A0, 1189
A1, 1189
A2, 1189
A3, 1189
A4, 1189
A5, 1189
A6, 1189
A7, 1189
A8, 1189
A9, 1189
B0, 1189
B1, 1189
B2, 1189
B3, 1189
B4, 1189
B5, 1189
B6, 1189
B7, 1189
B8, 1189
B9, 1189
begin_job, 1190
begin_page, 1190
end_job, 1190
end_page, 1191
EXECUTIVE, 1189
FOLIO, 1189
LANDSCAPE, 1190
LEDGER, 1189
LEGAL, 1189
LETTER, 1189
margins, 1191
ORIENTATION, 1190
Page_Format, 1189
Page_Layout, 1189
PORTRAIT, 1190
REVERSED, 1190
rotate, 1191
scale, 1191
start_job, 1192
start_page, 1192
TABLEOID, 1189
Fl_Paged_Device.cxx
  2268
Fl_Paged_Device.H
  2112
Fl_Paged_Device::page_format
  1959
fl_parse_color
  Fl_get_system_colors.cxx, 2260
fl_password
  Common Dialog Classes and Functions, 376
FL_PASTE
  Enumerations.H, 1985
FL_PATCH_VERSION
  Enumerations.H, 1977
fl_pie
  Drawing functions, 330
Fl_Pixmap
  Fl_Pixmap, 1192
color_average, 1196
copy, 1196

Generated by Doxygen
FL_PUSH
  Enumerations.H, 1983
fl_push_clip
  Drawing functions, 331
fl_push_matrix
  Drawing functions, 331
fl_putenv
  Unicode and UTF-8 functions, 352
FI_Radio_Button, 1265
  FI_Radio_Button, 1272
FI_Radio_Button.H, 2123
FI_Radio_Light_Button, 1272
FI_Radio_Light_Button.H, 2124
FI_Radio_Round_Button, 1280
  FI_Radio_Round_Button, 1287
FI_Radio_Round_Button.H, 2124
FL_READ
  Enumerations.H, 1979
fl_read_image
  Drawing functions, 331
FL_REASON_CANCELLED
  Enumerations.H, 1981
FL_REASON_CHANGED
  Enumerations.H, 1982
FL_REASON_CLOSED
  Enumerations.H, 1981
FL_REASON_DESELECTED
  Enumerations.H, 1981
FL_REASON_DRAGGED
  Enumerations.H, 1981
FL_REASON_ENTER_KEY
  Enumerations.H, 1982
FL_REASON_GOT_FOCUS
  Enumerations.H, 1982
FL_REASON_LOST_FOCUS
  Enumerations.H, 1982
FL_REASON_OPENED
  Enumerations.H, 1981
FL_REASON_RELEASED
  Enumerations.H, 1982
FL_REASON_RESELECTED
  Enumerations.H, 1981
FL_REASON_SELECTED
  Enumerations.H, 1981
FL_REASON_UNKNOWN
  Enumerations.H, 1981
FL_REASON_USER
  Enumerations.H, 1982
FI_Rect, 1288
  b, 1289
  FI_Rect, 1289
  inset, 1289
  r, 1290
fl_rect
  Drawing functions, 332
fl_rect.cxx, 2269
FI_Rect.H, 2125
fl_rectbound
fl_boxtype.cxx, 2253
fl_rectf
  Drawing functions, 332, 333
Fl_Widget
  platform_types.h, 2238
fl_register_images
  Fl_Shared_Image.H, 2133
FL_RELEASE
  Enumerations.H, 1983
fl_rename
  Unicode and UTF-8 functions, 352
FI_Repeat_Button, 1291
  FI_Repeat_Button, 1298
  handle, 1298
FI_Repeat_Button.H, 2126
fl_rescale_offscreen
  Drawing functions, 333
FL_RESERVED_TYPE
  FI_Widget.H, 2199
fl_reset_spot
  Drawing functions, 333
fl_restore_scale
  Drawing functions, 333
FI_Return_Button, 1299
  draw, 1307
  FI_Return_Button, 1306
  handle, 1307
FI_Return_Button.H, 2126
FI_RGB_Image, 1308
  array, 1314
  as_svg_image, 1312
  color_average, 1312
  copy, 1312
  desaturate, 1313
  draw, 1313
  FI_RGB_Image, 1311, 1312
  label, 1313
  max_size, 1314
  normalize, 1314
  uncache, 1314
FI_RGB_Image.H, 2127
FI_RGB_Scaling
  FI_Image.H, 2088
FI_RGB_SCALING_BILINEAR
  FI_Image.H, 2088
FI_RGB_SCALING_NEAREST
  FI_Image.H, 2088
fl_rmdir
  Unicode and UTF-8 functions, 353
FI_Roller, 1315
  draw, 1323
  FI_Roller, 1322
  handle, 1323
FI_Roller.H, 2127
fl_rotate
  Drawing functions, 334
FI_Round_Button, 1324
  FI_Round_Button, 1331
Fl_Round_Button.H, 2127
Fl_Round_Clock, 1332
  Fl_Round_Clock, 1339
Fl_Round_Clock.H, 2128
fl_rounded_rect
  Drawing functions, 334
fl_rounded_rectf
  Drawing functions, 334
fl_scale
  Drawing functions, 334
Fl_Scheme, 1339
  add_scheme_name, 1340
  names, 1341
  num_schemes, 1341
Fl_Scheme.H, 2128
Fl_Scheme_Choice, 1341
  Fl_Scheme_Choice, 1350
  handle, 1350
  init_value, 1351
  scheme_cb_, 1351
Fl_Scheme_Choice.H, 2129
FL_SCREEN_CONFIGURATION_CHANGED
  Enumerations.H, 1986
Fl_Screen_Driver.H, 2269
Fl_Scroll, 1351
  ~Fl_Scroll, 1361
  bbox, 1362
  delete_child, 1362
  draw, 1362
  fix_scrollbar_order, 1362
Fl_Scroll, 1361
  handle, 1363
  on_insert, 1363
  on_move, 1364
  recalc_scrollbars, 1364
  resize, 1365
  scroll_to, 1365
  scrollbar_size, 1365
  xposition, 1366
  yposition, 1366
fl_scroll
  Drawing functions, 335
Fl_Scroll.H, 2129
Fl_Scroll::Fl_Region_LRTB, 1290
Fl_Scroll::Fl_Region_XYWH, 1290
Fl_Scroll::Fl_Scrollbar_Data, 1377
Fl_Scroll::ScrollInfo, 1961
Fl_Scrollbar, 1366
  draw, 1375
  Fl_Scrollbar, 1375
  handle, 1375
  linesize, 1376
  value, 1376
Fl_Scrollbar.H, 2131
Fl_Secret_Input, 1377
  Fl_Secret_Input, 1387
  handle, 1387
Fl_Secret_Input.H, 2131
Fl_Select_Browser, 1388
  Fl_Select_Browser, 1401
Fl_Select_Browser.H, 2132
FL_SELECTIONCLEAR
  Enumerations.H, 1985
fl_set_spot
  Drawing functions, 335
fl_set_status
  Drawing functions, 335
Fl_Shared_Handler
  Fl_Shared_Image.H, 2133
Fl_Shared_Image, 1401
  ~Fl_Shared_Image, 1406
  add, 1406
  add_handler, 1406
  as_shared_image, 1406
  color_average, 1406
  compare, 1407
  copy, 1407, 1408
  copy_, 1408
  desaturate, 1408
  draw, 1409
  find, 1409
  Fl_Shared_Image, 1405
  get, 1409, 1410
  image, 1410
  images, 1411
  num_images, 1411
  original, 1411
  refcount, 1411
  release, 1412
  uncache, 1412
  update, 1412
Fl_Shared_Image.H, 2132, 2134
fl_register_images, 2133
Fl_Shared_Handler, 2133
FL_SHORTCUT
  Enumerations.H, 1985
Fl_Shortcut
  fl_types.h, 2190
Fl_Shortcut_Button, 1412
  draw, 1420
  Fl_Shortcut_Button, 1420
  handle, 1420
  value, 1420
Fl_Shortcut_Button.H, 2135
fl_shortcut_label
  Drawing functions, 335, 336
Fl_SHOW
  Enumerations.H, 1985
fl_show_colormap
  Color & Font functions, 302
fl_show_colormap.H, 2135, 2136
fl_show_input.H, 2136
Fl_Simple_Counter, 1421
Fl_Simple_Counter.H, 2136
Fl_Single_Window, 1429
  show, 1441
INDEX

Fl_Single_Window.H, 2137
fl_size
    Color & Font functions, 302
Fl_Slider, 1441
    bounds, 1450
draw, 1450
Fl_Slider, 1450
handle, 1450
scrollvalue, 1451
slider_size, 1451
Fl_Slider.H, 2137
FL_SOLID
    Drawing functions, 311
Fl_Spinner, 1451
draw, 1461
Fl_Spinner, 1461
handle, 1461
resize, 1462
step, 1462
type, 1463
value, 1463
wrap, 1463
Fl_Spinner.H, 2138
Fl_Spinner::Fl_Spinner_Input, 1464
handle, 1473

fl_stat
    Unicode and UTF-8 functions, 353
fl_strdup
    String handling functions, 361
Fl_String.H, 2272
fl_string_functions.h, 2140
FL_SUBMENU
    Fl_Menu_Item.H, 2102
FL_SUBMENU_POINTER
    Fl_Menu_Item.H, 2102
Fl_Surface_Device, 1474
    end_current, 1475
is_current, 1475
pop_current, 1475
push_current, 1475
set_current, 1476
surface, 1476
Fl_SVG_File_Surface, 1476
    ~Fl_SVG_File_Surface, 1479
close, 1479
Fl_SVG_File_Surface, 1478
origin, 1479
printable_rect, 1480
translate, 1480
untranslate, 1480
Fl_SVG_File_Surface.H, 2140
Fl_SVG_Image, 1480
    as_svg_image, 1486
color_average, 1486
copy, 1486
desaturate, 1486
draw, 1487
    Fl_SVG_Image, 1485
normalize, 1487
proportional, 1487
resize, 1487
Fl_SVG_Image.H, 2141
FL_SYMBOL_LABEL
    Enumerations.H, 1977
Fl_Sys_Menu_Bar, 1487
    about, 1498
add, 1499
clear, 1500
clear_submenu, 1500
create_window_menu, 1500
draw, 1500
Fl_Sys_Menu_Bar.H, 2142
Fl_Sys_Menu_Bar::Fl_Sys_Menu_Bar, 1498
insert, 1500, 1501
menu, 1501
mode, 1501
no_window_menu, 1498
remove, 1502
replace, 1502	
tabbing_mode_automatic, 1498
tabbing_mode_none, 1498
tabbing_mode_preferred, 1498
update, 1502
window_menu_style, 1502
window_menu_style_enum, 1498
Fl_Sys_Menu_Bar.H, 2142
Fl_Sys_Menu_Bar_Driver.H, 2273
fl_system
    Unicode and UTF-8 functions, 353
Fl_Sys_Menu_Bar_Driver.H, 2273
Fl_Table, 1503
    ~Fl_Table, 1518
array, 1518
callback, 1518
callback_col, 1519
callback_context, 1519
callback_row, 1519
child, 1519
children, 1520
clear, 1520
col_header, 1520
col_resize, 1520
col_resize_min, 1520
col_width, 1520
col_width_all, 1520
CONTEXT_CELL, 1518
CONTEXT_COL_HEADER, 1518
CONTEXT_ENDPAGE, 1518
CONTEXT_NONE, 1518
CONTEXT_RC_RESIZE, 1518
CONTEXT_ROW_HEADER, 1518
CONTEXT_STARTPAGE, 1518
CONTEXT_TABLE, 1518
cursor2rowcol, 1521
damage_zone, 1521
do_callback, 1521
draw, 1521

Generated by Doxygen
draw_cell, 1521
find_cell, 1523
Fl_Table, 1518
get_selection, 1523
handle, 1523
init_sizes, 1523
insert, 1524
is_interactive_resize, 1524
is_selected, 1524
move_cursor, 1524
recalc_dimensions, 1524
redraw_range, 1524
resize, 1525
row_col_clamp, 1525
row_header, 1525
row_height, 1525
row_height_all, 1525
row_resize, 1525
row_resize_min, 1525
rows, 1526
scrollbar_size, 1526
set_selection, 1526
tab_cell_nav, 1527
table_box, 1527
table_resized, 1527
table_scrolled, 1527
TableContext, 1517
top_row, 1528
visible_cells, 1528
when, 1528
Fl_Table.H, 2143
Fl_Table_Row, 1529
~Fl_Table_Row, 1542
clear, 1543
Fl_Table_Row, 1542
handle, 1543
row_selected, 1543
rows, 1543
select_all_rows, 1543
select_row, 1543
type, 1543
Fl_Table_Row.H, 2149
Fl_Tabs, 1544
clear_tab_positions, 1557
client_area, 1557
draw, 1558
draw_tab, 1558
Fl_Tabs, 1556
handle, 1558
handle_overflow, 1558
handle_overflow_menu, 1559
hit_close, 1559
hit_overflow_menu, 1559
hit_tabs_area, 1561
on_insert, 1561
on_move, 1561
on_remove, 1561
OVERFLOW_CLIP, 1556
OVERFLOW_COMPRESS, 1556
OVERFLOW_DRAG, 1556
OVERFLOW_PULLDOWN, 1556
overflow_type, 1564
push, 1561, 1562
redraw_tabs, 1562
resize, 1562
tab_align, 1562
tab_count, 1564
tab_flags, 1565
tab_height, 1562
tab_pos, 1565
tab_positions, 1563
tab_width, 1565
value, 1563, 1564
which, 1564
Fl_Tabs.H, 2151
Fl_Terminal, 1565
_RESERVED_1, 1586
_RESERVED_2, 1586
~Fl_Terminal, 1587
ansi, 1588
append, 1588
append_ascii, 1589
append_utf8, 1589
Attrib, 1585
BG_XTERM, 1586
BOLD, 1585
box, 1589
CharFlags, 1586
clear, 1589
clear_screen, 1589
clear_screen_home, 1590
color, 1590
CR_TO_LF, 1586
cursor_col, 1590
cursor_cr, 1590
cursor_down, 1590
cursor_right, 1591
cursor_row, 1591
cursor_up, 1591
delete_rows, 1591
DIM, 1585
display_columns, 1591
display_rows, 1592
draw, 1592
draw_buff, 1592
draw_row, 1592
draw_row_bg, 1592
EOL, 1586
FG_XTERM, 1586
Fl_Terminal, 1587
get_selection, 1592
h_to_row, 1592
handle, 1593
handle_unknown_char, 1593
history_lines, 1594
history_use, 1594
INDEX

kf_c_s_move, 1686
kf_copy, 1687
kf_ctrl_move, 1687
kf_cut, 1687
kf_default, 1687
kf_delete, 1687
kf_down, 1687
kf_end, 1688
kf_enter, 1688
kf_home, 1688
kf_ignore, 1688
kf_insert, 1688
kf_left, 1688
kf_m_s_move, 1688
kf_meta_move, 1689
kf_move, 1689
kf_page_down, 1689
kf_page_up, 1689
kf_paste, 1689
kf_redo, 1689
kf_right, 1690
kf_select_all, 1690
kf_shift_move, 1690
kf_undo, 1690
kf_up, 1690
remove_key_binding, 1690
tab_nav, 1690, 1691
Fl_Text_Editor.H, 2170
Fl_Text_Editor::Key_Binding, 1957
fl_text_extents
  Color & Font functions, 302, 303
Fl_Text_Selection, 1692
  end, 1693
  includes, 1693
  length, 1693
  position, 1693
  selected, 1694
  set, 1695
  start, 1695
  update, 1695
FL_THIN_DOWN_BOX
  Enumerations.H, 1979
FL_THIN_DOWN_FRAME
  Enumerations.H, 1979
FL_THIN_UP_BOX
  Enumerations.H, 1979
FL_THIN_UP_FRAME
  Enumerations.H, 1979
Fl_Tile, 1695
  cursor, 1707
  drag_intersection, 1707
  Fl_Tile, 1706
  handle, 1707
  init_size_range, 1708
  move_intersection, 1708
  on_insert, 1708
  on_move, 1708
  on_remove, 1708
  position, 1709
  requestGrowBottom, 1709
  requestGrowLeft, 1709
  requestGrowRight, 1709
  requestGrowTop, 1710
  requestShrinkBottom, 1710
  requestShrinkLeft, 1710
  requestShrinkRight, 1711
  requestShrinkTop, 1711
  resize, 1711
  set_cursor, 1712
  size_range, 1712
Fl_Tile.H, 2171
Fl_Tile::Size_Range, 1963
Fl_Tiled_Image, 1713
  color_average, 1716
  copy, 1716
  desaturate, 1716
  draw, 1716
  Fl_Tiled_Image, 1715
Fl_Tiled_Image.H, 2172
Fl_Timeout, 1717
  add_timeout, 1719
  current, 1720
  current_timeout, 1723
  elapsed_timeouts, 1720
  first_timeout, 1723
  free_timeout, 1723
  get, 1720
  has_timeout, 1721
  insert, 1721
  make_current, 1721
  release, 1722
  remove_timeout, 1722
  repeat_timeout, 1722
  time_to_wait, 1722
Fl_Timeout.cxx, 2276
Fl_Timeout.h, 2276, 2277
Fl_Timeout_Handler
  Callback Function Typedefs, 266
Fl_Timer, 1724
  direction, 1731
  draw, 1731
  Fl_Timer, 1730
  handle, 1731
  suspended, 1731
Fl_Timer.H, 2173
Fl_Timestamp
  platform_types.h, 2238
Fl_Toggle_Button, 1732
  Fl_Toggle_Button, 1739
Fl_Toggle_Button.H, 2173
Fl_Toggle_Light_Button.H, 2174
Fl_Toggle_Round_Button.H, 2174
Fl_Tooltip, 1740
  color, 1741
  current, 1741
  delay, 1742

Generated by Doxygen
disable, 1742
enable, 1742
enabled, 1742
enter_area, 1742
font, 1743
hidedelay, 1743
hoverdelay, 1743
margin_height, 1743
margin_width, 1743, 1744
size, 1744
textcolor, 1744
wrap_width, 1744
Fl_Tooltip.H, 2174
fl_transform_dx
  Drawing functions, 337
fl_transform_dy
  Drawing functions, 337
fl_transform_x
  Drawing functions, 337
fl_transform_y
  Drawing functions, 337
fl_transformed_vertex
  Drawing functions, 337
fl_translate
  Drawing functions, 338
Fl_Tree, 1745
  add, 1763, 1764
calc_dimensions, 1764
calc_tree, 1764
callback_item, 1765
callback_reason, 1765
clear, 1765
clear_children, 1765
close, 1766
closeicon, 1767
connectorstyle, 1767
deselect, 1767, 1768
deselect_all, 1768
display, 1769
displayed, 1769
draw, 1769
extend_selection, 1769
extend_selection_dir, 1770
find_clicked, 1770
find_item, 1771
first, 1771
first_selected_item, 1771
first_visible, 1772
first_visible_item, 1772
get_selected_items, 1772
handle, 1773
hposition, 1773
insert, 1773
insert_above, 1774
is_close, 1775
is_hscroll_visible, 1775
is_open, 1775, 1776
is_scrollbar, 1776
is_selected, 1777
is_vscroll_visible, 1777
item_clicked, 1777
item_draw_mode, 1778
item_labellbgcolor, 1778
item_labelbgcolor, 1778
item_labelbgcolor, 1778
item_labelfont, 1779
item_labelsize, 1779
item_pathname, 1779
item_reselect_mode, 1779
last, 1780
last_selected_item, 1780
last_visible, 1780
last_visible_item, 1780
load, 1781
next, 1781
next_item, 1781
next_selected_item, 1782
next_visible_item, 1783
open, 1783, 1784
open_toggle, 1785
openicon, 1785
prev, 1785
recalc_tree, 1786
remove, 1786
resize, 1786
root, 1786
root_label, 1787
scrollbar_size, 1787
select, 1787, 1788
select_all, 1788
select_only, 1789
select_toggle, 1789
selectbox, 1790
selectmode, 1790
set_item_focus, 1790
show_item, 1790, 1791
show_item_bottom, 1791
show_item_middle, 1791
show_item_top, 1791
show_self, 1792
showcollapse, 1792
showroot, 1792
sortorder, 1792
usericon, 1792, 1793
vposition, 1793
Fl_Tree.H, 2175, 2176
Fl_Tree_Reason, 2176
FL_TREE_REASON_CLOSED, 2176
FL_TREE_REASON_DESELECTED, 2176
FL_TREE_REASON_DRAGGED, 2176
FL_TREE_REASON_NONE, 2176
FL_TREE_REASON_OPENED, 2176
FL_TREE_REASON_RESELECTED, 2176
FL_TREE_REASON_SELECTED, 2176
Fl_Tree_Connector
  Fl_Tree_Prefs.H, 2185
FL_TREE_CONNECTOR_DOTTED
FL_TREE_SELECTABLE_ALWAYS, 2186
FL_TREE_SELECTABLE_ONCE, 2186
FL_Tree_Sort, 2186
FL_TREE_SORT_ASCENDING, 2186
FL_TREE_SORT_DESCENDING, 2186
FL_TREE_SORT_NONE, 2186
FL_Tree_Reason
   FL_Tree.H, 2176
FL_TREE_REASON_CLOSED
   FL_Tree.H, 2176
FL_TREE_REASON_DESELECTED
   FL_Tree.H, 2176
FL_TREE_REASON_DRAGGED
   FL_Tree.H, 2176
FL_TREE_REASON_NONE
   FL_Tree.H, 2176
FL_TREE_REASON_OPENED
   FL_Tree.H, 2176
FL_TREE_REASON_RESELECTED
   FL_Tree.H, 2176
FL_TREE_REASON_SELECTED
   FL_Tree.H, 2176
FL_Tree_Select
   FL_Tree_Prefs.H, 2186
FL_TREE_SELECT_MULTI
   FL_Tree_Prefs.H, 2186
FL_TREE_SELECT_NONE
   FL_Tree_Prefs.H, 2186
FL_TREE_SELECT_SINGLE
   FL_Tree_Prefs.H, 2186
FL_TREE_SELECT_SINGLE_DRAGGABLE
   FL_Tree_Prefs.H, 2186
FL_TREE_SELECTABLE_ALWAYS
   FL_Tree_Prefs.H, 2186
FL_TREE_SELECTABLE_ONCE
   FL_Tree_Prefs.H, 2186
FL_Tree_Sort
   FL_Tree_Prefs.H, 2186
   FL_Tree.H, 2176
FL_TREE_SORT_ASCENDING
   FL_Tree_Prefs.H, 2186
FL_TREE_SORT_DESCENDING
   FL_Tree_Prefs.H, 2186
FL_TREE_SORT_NONE
   FL_Tree_Prefs.H, 2186
fl_types.h, 2190
   FL_Shortcut, 2190
   fl_ucs_to_Utf16
   Unicode and UTF-8 functions, 354
   fl_uintptr_t
   platform_types.h, 2238
FL_UNFOCUS
   Enumerations.H, 1984
fl_unlink
   Unicode and UTF-8 functions, 354
FL_UP_BOX
   Enumerations.H, 1979
FL_UP_FRAME
   Enumerations.H, 1979
fi_utf8.h, 2191, 2193
   fi_utf8back
   Unicode and UTF-8 functions, 354
   fi_utf8bytes
   Unicode and UTF-8 functions, 354
   fi_utf8decode
   Unicode and UTF-8 functions, 355
   fi_utf8encode
   Unicode and UTF-8 functions, 355
   fi_utf8from_mb
   Unicode and UTF-8 functions, 355
   fi_utf8froma
   Unicode and UTF-8 functions, 356
   fi_utf8fromwc
   Unicode and UTF-8 functions, 356
   fi_utf8fwd
   Unicode and UTF-8 functions, 356
   fi_utf8len
   Unicode and UTF-8 functions, 357
   fi_utf8len1
   Unicode and UTF-8 functions, 357
   fi_utf8locale
   Unicode and UTF-8 functions, 357
   fi_utf8strlen
   Unicode and UTF-8 functions, 357
   fi_utf8stralen
   Unicode and UTF-8 functions, 357
   fi_utf8test
   Unicode and UTF-8 functions, 357
   fi_utf8to_mb
   Unicode and UTF-8 functions, 358
   fi_utf8toa
   Unicode and UTF-8 functions, 358
   fi_utf8toUtf16
   Unicode and UTF-8 functions, 358
   fl_utf8towc
   Unicode and UTF-8 functions, 359
   fl_utf8to_char
   Unicode and UTF-8 functions, 359
   fl_utf_strcasecmp
   Unicode and UTF-8 functions, 359
   fl_utf_strcasecmp
   Unicode and UTF-8 functions, 359
   fl_utf_strcmp
   Unicode and UTF-8 functions, 360
   fl_utf_tolower
   Unicode and UTF-8 functions, 360
   fl_utf_toupper
   Unicode and UTF-8 functions, 360
Fl_Valuator, 1823
   Fl_Valuator, 1831
   format, 1831
   increment, 1831
   maximum, 1832
   minimum, 1832
   precision, 1832
   range, 1832
   round, 1832
   step, 1833
   value, 1833
   value_damage, 1833
Fl_Valuator.H, 2195
Fl_Value_Input, 1833
cursor_color, 1842
draw, 1842
Fl_Value_Input, 1842
handle, 1842
resize, 1843
shortcut, 1843
soft, 1844
textcolor, 1844
textfont, 1844
textsize, 1844

Fl_Value_Input.H, 2196

Fl_Value_Output, 1845
draw, 1853
Fl_Value_Output, 1853
handle, 1853
soft, 1854
textcolor, 1854
textfont, 1854
textsize, 1854

Fl_Value_Output.H, 2197

Fl_Value_Slider, 1855
draw, 1864
Fl_Value_Slider, 1863
handle, 1864
value_height, 1864
value_width, 1865

Fl_Value_Slider.H, 2198

FL_VERSION
Enumerations.H, 1978

fl_vertex
Drawing functions, 338

fl_vertex.cxx, 2278

fl_vsnprintf
vsnprintf.c, 2293

fl_wcwidth
Unicode and UTF-8 functions, 360

fl_wcwidth_
Unicode and UTF-8 functions, 361

Fl_When
Enumerations.H, 1987

FL_WHEN_CHANGED
Enumerations.H, 1987

FL_WHEN_CLOSED
Enumerations.H, 1987

FL_WHEN_ENTER_KEY
Enumerations.H, 1987

FL_WHEN_ENTER_KEY_ALWAYS
Enumerations.H, 1987

FL_WHEN_ENTER_KEY_CHANGED
Enumerations.H, 1987

FL_WHEN_NEVER
Enumerations.H, 1987

FL_WHEN_NOT_CHANGED
Enumerations.H, 1987

FL_WHEN_RELEASE
Enumerations.H, 1987

FL_WHEN_RELEASE_ALWAYS
Enumerations.H, 1987

~Fl_Widget, 1874
activate, 1874
active, 1874
active_r, 1874
align, 1874, 1875
argument, 1875
as_gl_window, 1875
as_group, 1876
as_window, 1876
AUTO_DELETE_USER_DATA, 1873
bind_deimage, 1877
bind_image, 1877, 1878
box, 1878
callback, 1878, 1879
CHANGED, 1873
changed, 1880
clear_active, 1880
clear_changed, 1880
clear_damage, 1880
clear_output, 1881
clear_visible, 1881
clear_visible_focus, 1881
CLIP_CHILDREN, 1873
color, 1881, 1882
color2, 1882
contains, 1882
COPIED_LABEL, 1873
COPIED_TOOLTIP, 1873
copy_label, 1882
copy_tooltip, 1883
damage, 1883, 1884
deactivate, 1884
default_callback, 1884
deimage, 1885
DEIMAGE_BOUND, 1873
deimage_bound, 1885
do_callback, 1886
draw, 1887
draw_focus, 1887, 1888
draw_label, 1888, 1889
Fl_Widget, 1873
FORCE_POSITION, 1872
FULLSCREEN, 1873
GROUP_RELATIVE, 1873
h, 1889
handle, 1889
hide, 1890
image, 1890
IMAGE_BOUND, 1873
image_bound, 1891
INACTIVE, 1872
inside, 1891
INVISIBLE, 1872
is_label_copied, 1891
label, 1891, 1892
label_shortcut, 1892

Generated by Doxygen
labelcolor, 1893
labelfont, 1893
labels, 1894
labelcolor, 1893
labelfont, 1893
labels, 1894
labeltype, 1894
MAC_USE_ACCENTS_MENU, 1873
MAXIMIZED, 1873
measure_label, 1894
MENU_WINDOW, 1873
MODAL, 1873
NEEDS_KEYBOARD, 1873
needs_keyboard, 1895
NOOVERLAY, 1873
NOBORDER, 1872
NON_MODAL, 1872
OUTPUT, 1872
output, 1895
OVERRIDE, 1873
parent, 1896
position, 1896
redraw, 1896
redraw_label, 1896
resize, 1896
selection_color, 1897
set_active, 1897
set_changed, 1898
set_output, 1898
set_visible, 1898
set_visible_focus, 1898
SHORTCUT_LABEL, 1872
shortcut_label, 1898
show, 1898
size, 1899
take_focus, 1899
takeevents, 1899
test_shortcut, 1899, 1900
tooltip, 1900
TOOLTIP_WINDOW, 1873
top_window, 1901
top_window_offset, 1901
type, 1901
user_data, 1902
USERFLAG1, 1873
USERFLAG2, 1873
USERFLAG3, 1873
visible, 1892
VISIBLE_FOCUS, 1873
visible_focus, 1902
visible_r, 1902
w, 1903
when, 1903
window, 1904
x, 1904
y, 1904, 1905
FI_Widget.H, 2199
FI_RESERVED_TYPE, 2199
FI_Widget_Surface, 1905
draw, 1907
draw_decorated_window, 1907
FI_Widget_Surface.H, 2204
FI_Widget_Tracker, 1909
deleted, 1910
exists, 1910
widget, 1910
fl_width
Color & Font functions, 303
FI_Window, 1910
~FI_Window, 1923
as_double_window, 1923
as_overlay_window, 1923
as_window, 1923
border, 1924
clear_border, 1924
clear_modal_states, 1924
current, 1924
current_, 1940
cursor, 1924, 1925
decorated_h, 1925
decorated_w, 1925
default_cursor, 1926
default_icon, 1926
default_icons, 1926, 1927
default_size_range, 1927
default_xclass, 1928
draw, 1929
FI_Window, 1922
flush, 1929
force_position, 1929
free_icons, 1929
free_position, 1930
fullscreen, 1930
fullscreen_screens, 1930
handle, 1930
hide, 1931
hotspot, 1931
icon, 1931, 1932
iconize, 1932
icons, 1932, 1933
isResizable, 1933
make_current, 1933
maximize, 1933
modal, 1934
os_id, 1934
resize, 1934
screen_num, 1934
set_menu_window, 1934
set_modal, 1935
set_non_modal, 1935
set_tooltip_window, 1935
shape, 1935
show, 1936
INDEX

Fl_Shared_Image, 1410
Fl_Widget, 1890

IMAGE_BOUND
Fl_Widget, 1873

image_bound
Fl_Widget, 1891

image_label
Fl_Menu_Item, 1083

images
Fl_Shared_Image, 1411
imKStoUCS.c, 2358

in_selection
Fl_Text_Display, 1650

INACTIVE
Fl_Widget, 1872

inactive
Fl_Image, 938

includes
Fl_Text_Selection, 1693

incr_height
Fl_Browser, 482
Fl_Browser_, 509

increment
Fl_Valuator, 1831

index
Fl_Input_, 974

init_size_range
Fl_Tile, 1708

init_sizes
Fl_Group, 857
Fl_Table, 1523

init_value
Fl.Scheme_Choice, 1351

inp_x
Fl_Input_Choice, 997

input
Fl_Input_Choice, 998

input_type
Fl_Input_, 974

insert
Fl_Browser, 482, 483
Fl_Chart, 555
Fl_Group, 857
Fl_Input_, 974
Fl_Menu_, 1048
Fl_Menu_Item, 1083
Fl_Sys_Menu_Bar, 1500, 1501
Fl_Table, 1524
Fl_Text_Buffer, 1613
Fl_Text_Display, 1650
Fl_Timeout, 1721
Fl_Tree, 1773
Fl_Tree_Item, 1805
Fl_Tree_Item_Array, 1816

insert
Fl_Text_Buffer, 1614

insert_above
Fl_Tree, 1774

Fl_Tree_Item, 1806
insert_char
Fl_Terminal, 1594
insert_mode
Fl_Text_Editor, 1686
insert_position
Fl_Input_, 975
Fl_Text_Display, 1651
insert_rows
Fl_Terminal, 1594
insertfile
Fl_Text_Buffer, 1614
inserting
Fl_Browser_, 509

inset
Fl_Rect, 1289

inside
Fl_Widget, 1891

Introduction to FLTK, 4

INVERSE
Fl_Terminal, 1586

INVISIBLE
Fl_Widget, 1872

is_animated
Fl_Anim_GIF_Image, 442

is_close
Fl_Tree, 1775

is_current
Fl_Copy_Surface, 626
Fl_Image_Surface, 945
Fl_Printer, 1254
Fl_Surface_Device, 1475

is_hscroll_visible
Fl_Tree, 1775

is_inside_selection
Fl_Terminal, 1594

is_interactive_resize
Fl_Table, 1524

is_label_copied
Fl_Widget, 1891

is_open
Fl_Tree, 1775, 1776

isResizable
Fl_Window, 1933

is_scheme
Fl, 410

is_scrollbar
Fl_Tree, 1776

is_selected
Fl_Table, 1524
Fl_Tree, 1777

is_vscroll_visible
Fl_Tree, 1777

is_word_separator
Fl_Text_Buffer, 1614

iso8859_1.h, 2523
iso8859_10.h, 2523
iso8859_11.h, 2524
<table>
<thead>
<tr>
<th>ISO8859 Related Files</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>iso8859_13.h</td>
<td>2525</td>
</tr>
<tr>
<td>iso8859_14.h</td>
<td>2526</td>
</tr>
<tr>
<td>iso8859_15.h</td>
<td>2528</td>
</tr>
<tr>
<td>iso8859_16.h</td>
<td>2528</td>
</tr>
<tr>
<td>iso8859_2.h</td>
<td>2530</td>
</tr>
<tr>
<td>iso8859_3.h</td>
<td>2531</td>
</tr>
<tr>
<td>iso8859_4.h</td>
<td>2532</td>
</tr>
<tr>
<td>iso8859_5.h</td>
<td>2533</td>
</tr>
<tr>
<td>iso8859_6.h</td>
<td>2534</td>
</tr>
<tr>
<td>iso8859_7.h</td>
<td>2535</td>
</tr>
<tr>
<td>iso8859_8.h</td>
<td>2536</td>
</tr>
<tr>
<td>iso8859_9.h</td>
<td>2537</td>
</tr>
<tr>
<td>iso8859_9e.h</td>
<td>2538</td>
</tr>
</tbody>
</table>

ITALIC

- `Fl_Terminal`, 1585
- `item_at`:
  - `Fl_Browser`, 483
  - `Fl_Browser`, 509
  - `Fl_Check_Browser`, 569
- `item_clicked`:
  - `Fl_Tree`, 1777
- `item_draw`:
  - `Fl_Browser`, 483
  - `Fl_Browser`, 510
  - `Fl_Check_Browser`, 569
- `item_draw_mode`:
  - `Fl_Tree`, 1778
  - `Fl_Tree_Prefs`, 1821
- `item_first`:
  - `Fl_Browser`, 484
  - `Fl_Browser`, 510
  - `Fl_Check_Browser`, 570
- `item_height`:
  - `Fl_Browser`, 484
  - `Fl_Browser`, 510
  - `Fl_Check_Browser`, 570
- `item_labelbgcolor`:
  - `Fl_Tree`, 1778
  - `Fl_Tree_Prefs`, 1821
- `item_labelbgcolor`:
  - `Fl_Tree`, 1778
- `item_labelfont`:
  - `Fl_Tree`, 1779
- `item_labelsize`:
  - `Fl_Tree`, 1779
- `item_last`:
  - `Fl_Browser`, 484
  - `Fl_Browser`, 510
- `item_next`:
  - `Fl_Browser`, 484
  - `Fl_Browser`, 510
  - `Fl_Check_Browser`, 570
- `item_pathname`:
  - `Fl_Menu`, 1049
  - `Fl_Tree`, 1779
- `item_prev`:
  - `Fl_Browser`, 485
  - `Fl_Browser`, 511
  - `Fl_Check_Browser`, 570

`Fl_Check_Browser`, 570
- `item_quick_height`:
  - `Fl_Browser`, 485
  - `Fl_Browser`, 510
  - `Fl_Check_Browser`, 570
- `item_reselect_mode`:
  - `Fl_Tree`, 1779
- `item_select`:
  - `Fl_Browser`, 485
  - `Fl_Browser`, 511
  - `Fl_Check_Browser`, 570
- `item_selected`:
  - `Fl_Browser`, 486
  - `Fl_Browser`, 511
  - `Fl_Check_Browser`, 571
- `item_swap`:
  - `Fl_Browser`, 486
  - `Fl_Browser`, 512
  - `Fl_Check_Browser`, 571
- `item_text`:
  - `Fl_Browser`, 486
  - `Fl_Browser`, 512
  - `Fl_Check_Browser`, 571
- `item_width`:
  - `Fl_Browser`, 486
  - `Fl_Browser`, 512
  - `Fl_Check_Browser`, 572
- `ivalue`:
  - `Fl_Input`, 976

- `jisx0201.h`, 2540
- `jisx0208.h`, 2540
- `jisx0212.h`, 2568

keyboard_screen_scaling

- `Screen functions`, 288

kf_backspace

- `Fl_Text_Editor`, 1686

kf_c_s_move

- `Fl_Text_Editor`, 1686

kf_copy

- `Fl_Text_Editor`, 1687

kf_ctrl_move

- `Fl_Text_Editor`, 1687

kf_cut

- `Fl_Text_Editor`, 1687

kf_default

- `Fl_Text_Editor`, 1687

kf_delete

- `Fl_Text_Editor`, 1687

kf_down

- `Fl_Text_Editor`, 1687

kf_end

- `Fl_Text_Editor`, 1688

kf_enter

- `Fl_Text_Editor`, 1688

kf_home

- `Fl_Text_Editor`, 1688

kf_ignore

- `Fl_Text_Editor`, 1688

kf_insert

Generated by Doxygen
_Fl_Text_Editor, 1688
kf_left
_Fl_Text_Editor, 1688
kf_m_s_move
_Fl_Text_Editor, 1688
kf_meta_move
_Fl_Text_Editor, 1689
kf_move
_Fl_Text_Editor, 1689
kf_page_down
_Fl_Text_Editor, 1689
kf_page_up
_Fl_Text_Editor, 1689
kf_paste
_Fl_Text_Editor, 1689
kf_redo
_Fl_Text_Editor, 1689
kf_right
_Fl_Text_Editor, 1690
kf_select_all
_Fl_Text_Editor, 1690
kf_shift_move
_Fl_Text_Editor, 1690
kf_undo
_Fl_Text_Editor, 1690
kf_up
_Fl_Text_Editor, 1690
koi8_c.h, 2593
koi8_r.h, 2594
koi8_u.h, 2596
ksc5601.h, 2598
label
_Fl_Bitmap, 449
_Fl_File_Icon, 692
_Fl_Image, 939
_Fl_Menu_Item, 1084, 1085
_Fl_Multi_Label, 1116
_Fl_Pixmap, 1197
_Fl_RGB_Img, 1313
_Fl_Tree_Item, 1806
_Fl_Widget, 1891, 1892
label_h
_Fl_Tree_Item, 1806
label_shortcut
_Fl_Widget, 1892
label_w
_Fl_Tree_Item, 1806
label_x
_Fl_Tree_Item, 1806
label_y
_Fl_Tree_Item, 1806
labela
_Fl_Multi_Label, 1117
labelb
_Fl_Multi_Label, 1117
labelbgcolor
_Fl_Tree_Item, 1807
labelcolor
_Fl_Menu_Item, 1085
_Fl_Widget, 1893
labelfont
_Fl_Menu_Item, 1085
_Fl_Widget, 1893
labelsize
_Fl_Widget, 1894
labeltype
_Fl_File_Icon, 693
_Fl_Menu_Item, 1086
_Fl_Widget, 1894
LANDSCAPE
_Fl_Paged_Device, 1190
last
_Fl_Tree, 1780
last_selected_item
_Fl_Tree, 1780
last_visible
_Fl_Tree, 1780
last_visible_item
_Fl_Tree, 1780
layout
_Fl_Flex, 737
_Fl_Grid, 838
ld
_Fl_Image, 939
LEDGER
_Fl_Paged_Device, 1189
leftedge
_Fl_Browser, 513
leftline
_Fl_Help_View, 878
LEGAL
_Fl_Paged_Device, 1189
length
_Fl_Text_Buffer, 1614
_Fl_Text_Selection, 1693
LETTER
_Fl_Paged_Device, 1189
LF_TO_CR
_Fl_Terminal, 1586
LF_TO_CRLF
_Fl_Terminal, 1586
line_end
_Fl_Input, 976
_Fl_Text_Buffer, 1614
_Fl_Text_Display, 1651
line_start
_Fl_Input, 976
_Fl_Text_Buffer, 1615
_Fl_Text_Display, 1651
line_text
_Fl_Text_Buffer, 1615
lineno
_Fl_Browser, 487
linenumber_align
_Fl_Text_Display, 1652
linenumberbgcolor
INDEX

Fl_Text_Display, 1652
linenumber_fgcolor
Fl_Text_Display, 1652
linenumber_font
Fl_Text_Display, 1652
linenumber_format
Fl_Text_Display, 1652
linenumber_size
Fl_Text_Display, 1653
linenumber_width
Fl_Text_Display, 1653
lineposition
Fl_Browser, 487
linesize
Fl_Scrollbar, 1376
linespacing
Fl_Browser, 513
link
Fl_Help_View, 879
load
Fl_Anim_GIF_Image, 442
Fl_Browser, 487
Fl_File_Browser, 682
Fl_File_Icon, 693
Fl_Help_Dialog, 864
Fl_Help_View, 879
Fl_Plugin_Manager, 1202
Fl_Tree, 1781
load_fti
Fl_File_Icon, 693
load_image
Fl_File_Icon, 693
load_system_icons
Fl_File_Icon, 694
loadAll
Fl_Plugin_Manager, 1202
loadfile
Fl_Text_Buffer, 1615
lock
Multithreading support functions, 339
LOG_FLAG
Fl_Anim_GIF_Image, 435
longest_vline
Fl_Text_Display, 1653
loop
Fl_Anim_GIF_Image, 444
lstep
Fl_Counter, 637
Mac OS X-specific symbols, 361
fl_mac_os_version, 362
fl_mac_set_about, 362
fl_open_callback, 362
mac.H, 2232
MAC_USE_ACCENTS_MENU
Fl_Widget, 1873
maintain_absolute_top_line_number
Fl_Text_Display, 1653
maintaining_absolute_top_line_number
Fl_Text_Display, 1654
make_current
Fl_Gl_Window, 801
Fl_Timeout, 1721
Fl_Window, 1933
make_overlay_current
Fl_Gl_Window, 801
make_visible
Fl_Browser, 488
manage_item_destroy
Fl_Tree_Item_Array, 1816
margin
Fl_Flex, 737, 738
Fl_Grid, 838, 839
margin_height
Fl_Tooltip, 1743
margin_width
Fl_Tooltip, 1743, 1744
marginbottom
Fl_Tree_Prefs, 1821
margins
Fl_Paged_Device, 1191
Fl_PostScript_File_Device, 1225
Fl_Printer, 1254
mark
Fl_Input_, 976, 977
mask
Fl_Image_Surface, 945
math.h, 2234
max_size
Fl_RGB_Image, 1314
maximize
Fl_Window, 1933
MAXIMIZED
Fl_Widget, 1873
maximum
Fl_Progress, 1264
Fl_Valuator, 1832
maximum_size
Fl_Input_, 977
maxsize
Fl_Chart, 556
measure
Fl_Label, 1014
Fl_Menu_Item, 1086
measure_deleted_lines
Fl_Text_Display, 1654
measure_label
Fl_Widget, 1894
measure_proportional_character
Fl_Text_Display, 1654
measure_vline
Fl_Text_Display, 1655
mediumarrow.h, 2289
MEMORY
Fl_Preferences, 1232
menu
Fl_Menu_, 1049, 1050

Generated by Doxygen
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Widget_Surface</td>
<td>1908</td>
</tr>
<tr>
<td>Fl_Copy_Surface</td>
<td>626</td>
</tr>
<tr>
<td>Fl_EPS_File_Surface</td>
<td>667</td>
</tr>
<tr>
<td>Fl_Image_Surface</td>
<td>946</td>
</tr>
<tr>
<td>Fl_PostScript_File_Device</td>
<td>1226</td>
</tr>
<tr>
<td>Fl_Printer</td>
<td>1255</td>
</tr>
<tr>
<td>Fl_SVG_File_Surface</td>
<td>1480</td>
</tr>
<tr>
<td>Fl_Widget_Surface</td>
<td>1908</td>
</tr>
<tr>
<td>printf</td>
<td></td>
</tr>
<tr>
<td>Fl_Terminal</td>
<td>1595</td>
</tr>
<tr>
<td>Fl_Text_Buffer</td>
<td>1617</td>
</tr>
<tr>
<td>program_should_quit</td>
<td></td>
</tr>
<tr>
<td>Fl, 412</td>
<td></td>
</tr>
<tr>
<td>Programming with FLUID, 97</td>
<td></td>
</tr>
<tr>
<td>proportional</td>
<td></td>
</tr>
<tr>
<td>Fl_SVG_Image</td>
<td>1487</td>
</tr>
<tr>
<td>pulldown</td>
<td></td>
</tr>
<tr>
<td>Fl_Menu_Item</td>
<td>1087</td>
</tr>
<tr>
<td>push</td>
<td></td>
</tr>
<tr>
<td>Fl_Tabs, 1561, 1562</td>
<td></td>
</tr>
<tr>
<td>push_current</td>
<td></td>
</tr>
<tr>
<td>Fl_Surface_Device</td>
<td>1475</td>
</tr>
<tr>
<td>pushed</td>
<td></td>
</tr>
<tr>
<td>Events handling functions, 280</td>
<td></td>
</tr>
<tr>
<td>putchar</td>
<td></td>
</tr>
<tr>
<td>Fl_Terminal</td>
<td>1595, 1596</td>
</tr>
<tr>
<td>r</td>
<td></td>
</tr>
<tr>
<td>Fl_Color_Chooser</td>
<td>622</td>
</tr>
<tr>
<td>Fl_Rect, 1290</td>
<td></td>
</tr>
<tr>
<td>radio</td>
<td></td>
</tr>
<tr>
<td>Fl_Menu_Item</td>
<td>1087</td>
</tr>
<tr>
<td>range</td>
<td></td>
</tr>
<tr>
<td>Fl_Valuator, 1832</td>
<td></td>
</tr>
<tr>
<td>RATE_LIMITED</td>
<td></td>
</tr>
<tr>
<td>Fl_Terminal</td>
<td>1586</td>
</tr>
<tr>
<td>readonly</td>
<td></td>
</tr>
<tr>
<td>Fl_Input, 978</td>
<td></td>
</tr>
<tr>
<td>readqueue</td>
<td></td>
</tr>
<tr>
<td>Fl, 413</td>
<td></td>
</tr>
<tr>
<td>ready</td>
<td></td>
</tr>
<tr>
<td>Fl, 413</td>
<td></td>
</tr>
<tr>
<td>recalc_dimensions</td>
<td></td>
</tr>
<tr>
<td>Fl_Table, 1524</td>
<td></td>
</tr>
<tr>
<td>recalc_scrollbars</td>
<td></td>
</tr>
<tr>
<td>Fl_Scroll, 1364</td>
<td></td>
</tr>
<tr>
<td>recalc_tree</td>
<td></td>
</tr>
<tr>
<td>Fl_Tree, 1786</td>
<td></td>
</tr>
<tr>
<td>Fl_Tree_Item</td>
<td>1810</td>
</tr>
<tr>
<td>rectangle_capture</td>
<td></td>
</tr>
<tr>
<td>Fl_Device_Plugin</td>
<td>639</td>
</tr>
<tr>
<td>redisplay_range</td>
<td></td>
</tr>
<tr>
<td>Fl_Text_Display</td>
<td>1658</td>
</tr>
<tr>
<td>redo</td>
<td></td>
</tr>
<tr>
<td>Fl_Input, 978</td>
<td></td>
</tr>
<tr>
<td>redo</td>
<td></td>
</tr>
<tr>
<td>Fl_Widget, 1896</td>
<td></td>
</tr>
<tr>
<td>redraw_line</td>
<td></td>
</tr>
<tr>
<td>Fl_Browser, 513</td>
<td></td>
</tr>
<tr>
<td>redraw_lines</td>
<td></td>
</tr>
<tr>
<td>Fl_Browser, 514</td>
<td></td>
</tr>
<tr>
<td>redraw_overlay</td>
<td></td>
</tr>
<tr>
<td>Fl_Gl_Window, 803</td>
<td></td>
</tr>
<tr>
<td>Fl_Overlay_Window</td>
<td>1175</td>
</tr>
<tr>
<td>redraw_range</td>
<td></td>
</tr>
<tr>
<td>Fl_Table, 1524</td>
<td></td>
</tr>
<tr>
<td>redraw_rate</td>
<td></td>
</tr>
<tr>
<td>Fl_Terminal</td>
<td>1596</td>
</tr>
<tr>
<td>redraw_style</td>
<td></td>
</tr>
<tr>
<td>Fl_Terminal</td>
<td>1596, 1597</td>
</tr>
<tr>
<td>redraw_tabs</td>
<td></td>
</tr>
<tr>
<td>Fl_Tabs, 1562</td>
<td></td>
</tr>
<tr>
<td>RedrawStyle</td>
<td></td>
</tr>
<tr>
<td>Fl_Terminal</td>
<td>1586</td>
</tr>
<tr>
<td>refcount</td>
<td></td>
</tr>
<tr>
<td>Fl_Shared_Image</td>
<td>1411</td>
</tr>
<tr>
<td>release</td>
<td></td>
</tr>
<tr>
<td>Fl, 413</td>
<td></td>
</tr>
<tr>
<td>Fl_Image, 939</td>
<td></td>
</tr>
<tr>
<td>Fl_Shared_Image</td>
<td>1412</td>
</tr>
<tr>
<td>Fl_Timeout, 1722</td>
<td></td>
</tr>
<tr>
<td>release_widget_pointer</td>
<td></td>
</tr>
<tr>
<td>Safe widget deletion support functions, 341</td>
<td></td>
</tr>
<tr>
<td>reload_scheme</td>
<td></td>
</tr>
<tr>
<td>Fl, 413</td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td></td>
</tr>
<tr>
<td>Fl_Browser, 490</td>
<td></td>
</tr>
<tr>
<td>Fl_Check_Browser, 572</td>
<td></td>
</tr>
<tr>
<td>Fl_Group, 859</td>
<td></td>
</tr>
<tr>
<td>Fl_Menu, 1051</td>
<td></td>
</tr>
<tr>
<td>Fl_Sys_Menu_Bar</td>
<td>1502</td>
</tr>
<tr>
<td>Fl_Text_Buffer</td>
<td>1618</td>
</tr>
<tr>
<td>Fl_Tree, 1786</td>
<td></td>
</tr>
<tr>
<td>Fl_Tree_Item_Array</td>
<td>1817</td>
</tr>
<tr>
<td>remove_</td>
<td></td>
</tr>
<tr>
<td>Fl_Text_Buffer, 1618</td>
<td></td>
</tr>
<tr>
<td>remove_check</td>
<td></td>
</tr>
<tr>
<td>Fl, 413</td>
<td></td>
</tr>
<tr>
<td>remove_child</td>
<td></td>
</tr>
<tr>
<td>Fl_Tree_Item, 1810, 1811</td>
<td></td>
</tr>
<tr>
<td>remove_handler</td>
<td></td>
</tr>
<tr>
<td>Events handling functions, 280</td>
<td></td>
</tr>
<tr>
<td>remove_icon</td>
<td></td>
</tr>
<tr>
<td>Fl_Browser, 490</td>
<td></td>
</tr>
<tr>
<td>remove_key_binding</td>
<td></td>
</tr>
<tr>
<td>Fl_Text_Editor, 1690</td>
<td></td>
</tr>
<tr>
<td>remove_system_handler</td>
<td></td>
</tr>
<tr>
<td>Events handling functions, 281</td>
<td></td>
</tr>
<tr>
<td>remove_timeout</td>
<td></td>
</tr>
<tr>
<td>Fl, 414</td>
<td></td>
</tr>
<tr>
<td>Fl_Timeout, 1722</td>
<td></td>
</tr>
<tr>
<td>removePlugin</td>
<td></td>
</tr>
<tr>
<td>Fl_Plugin_Manager, 1204</td>
<td></td>
</tr>
<tr>
<td>reparent</td>
<td></td>
</tr>
<tr>
<td>Term</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Fl_Tree_Item</td>
<td>1811</td>
</tr>
<tr>
<td>Fl_Tree_Item_Array</td>
<td>1817</td>
</tr>
<tr>
<td>repeat_timeout</td>
<td>414</td>
</tr>
<tr>
<td>Fl_Timeout</td>
<td>1722</td>
</tr>
<tr>
<td>replace</td>
<td>556</td>
</tr>
<tr>
<td>Fl_Chart</td>
<td>978</td>
</tr>
<tr>
<td>Fl_Menu_</td>
<td>1052</td>
</tr>
<tr>
<td>Fl_Sys_Menu_Bar</td>
<td>1502</td>
</tr>
<tr>
<td>Fl_Text_Buffer</td>
<td>1618</td>
</tr>
<tr>
<td>Fl_Tree_Item</td>
<td>1811</td>
</tr>
<tr>
<td>Fl_Tree_Item_Array</td>
<td>1817</td>
</tr>
<tr>
<td>replace_child</td>
<td>1812</td>
</tr>
<tr>
<td>replacing</td>
<td>514</td>
</tr>
<tr>
<td>request_grow_b</td>
<td>1709</td>
</tr>
<tr>
<td>request_grow_l</td>
<td>1709</td>
</tr>
<tr>
<td>request_grow_r</td>
<td>1709</td>
</tr>
<tr>
<td>request_grow_t</td>
<td>1710</td>
</tr>
<tr>
<td>request_shrink_b</td>
<td>1710</td>
</tr>
<tr>
<td>request_shrink_l</td>
<td>1710</td>
</tr>
<tr>
<td>request_shrink_r</td>
<td>1711</td>
</tr>
<tr>
<td>request_shrink_t</td>
<td>1711</td>
</tr>
<tr>
<td>rescale</td>
<td>947</td>
</tr>
<tr>
<td>reset_absolute_top_line_number</td>
<td>1658</td>
</tr>
<tr>
<td>reset_terminal</td>
<td>1597</td>
</tr>
<tr>
<td>resizable</td>
<td>859</td>
</tr>
<tr>
<td>resize</td>
<td>860</td>
</tr>
<tr>
<td>Fl_Anim_GIF_Image</td>
<td>443</td>
</tr>
<tr>
<td>Fl_Browser_</td>
<td>514</td>
</tr>
<tr>
<td>Fl_Double_Window</td>
<td>662</td>
</tr>
<tr>
<td>Fl_Flex</td>
<td>739</td>
</tr>
<tr>
<td>Fl_Gl_Window</td>
<td>803</td>
</tr>
<tr>
<td>Fl_Grid</td>
<td>840</td>
</tr>
<tr>
<td>Fl_Group</td>
<td>861</td>
</tr>
<tr>
<td>Fl_Help_View</td>
<td>879</td>
</tr>
<tr>
<td>Fl_Input_</td>
<td>979</td>
</tr>
<tr>
<td>Fl_Input_Choice</td>
<td>998</td>
</tr>
<tr>
<td>Fl_Overlay_Window</td>
<td>1175</td>
</tr>
<tr>
<td>Fl_Pack</td>
<td>1186</td>
</tr>
<tr>
<td>Fl_Scroll</td>
<td>1365</td>
</tr>
<tr>
<td>Fl_Spinner</td>
<td>1462</td>
</tr>
<tr>
<td>Fl_SVG_Image</td>
<td>1487</td>
</tr>
<tr>
<td>Fl_Table</td>
<td>1525</td>
</tr>
<tr>
<td>Fl_Tabs</td>
<td>1562</td>
</tr>
<tr>
<td>Fl_Terminal</td>
<td>1597</td>
</tr>
<tr>
<td>Fl_Text_Display</td>
<td>1658</td>
</tr>
<tr>
<td>Fl_Tile</td>
<td>1711</td>
</tr>
<tr>
<td>Fl_Tree</td>
<td>1786</td>
</tr>
<tr>
<td>Fl_Value_Input</td>
<td>1843</td>
</tr>
<tr>
<td>Fl_Widget</td>
<td>1896</td>
</tr>
<tr>
<td>Fl_Window</td>
<td>1934</td>
</tr>
<tr>
<td>REVERSED</td>
<td>1232</td>
</tr>
<tr>
<td>root</td>
<td>1786</td>
</tr>
<tr>
<td>root_label</td>
<td>1787</td>
</tr>
<tr>
<td>ROOT_MASK</td>
<td>1232</td>
</tr>
<tr>
<td>rotate</td>
<td>1191</td>
</tr>
<tr>
<td>Fl_Paged_Device</td>
<td>1226</td>
</tr>
<tr>
<td>Fl_Printer</td>
<td>1255</td>
</tr>
<tr>
<td>round</td>
<td>1832</td>
</tr>
<tr>
<td>ROW</td>
<td>733</td>
</tr>
<tr>
<td>row_col_clamp</td>
<td>1525</td>
</tr>
<tr>
<td>row_gap</td>
<td>840</td>
</tr>
<tr>
<td>row_header</td>
<td>1525</td>
</tr>
<tr>
<td>row_height</td>
<td>840</td>
</tr>
<tr>
<td>row_height_all</td>
<td>1525</td>
</tr>
<tr>
<td>row_resize</td>
<td>1525</td>
</tr>
<tr>
<td>row_resize_min</td>
<td>1525</td>
</tr>
<tr>
<td>row_selected</td>
<td>1543</td>
</tr>
<tr>
<td>rows</td>
<td>841</td>
</tr>
<tr>
<td>run</td>
<td>1526</td>
</tr>
<tr>
<td>Fl_Table_Row</td>
<td>1543</td>
</tr>
</tbody>
</table>
INDEX

Fl, 414

Safe widget deletion support functions, 340
  clear_widget_pointer, 341
  delete_widget, 341
  do_widget_deletion, 341
  release_widget_pointer, 341
  watch_widget_pointer, 342

saturation
  Fl_Color_Chooser, 623

SAVEAS_CONFIRM
  Fl_Native_File_Chooser, 1140

savefile
  Fl_Text_Buffer, 1618

scale
  Fl_Image, 940
  Fl_Paged_Device, 1191
  Fl_PostScript_File_Device, 1226
  Fl_Printer, 1256

scaling_algorithm
  Fl_Image, 940

scheme
  Fl, 415
  scheme_cb_
    Fl_Scheme_Choice, 1351

Screen functions, 287
  keyboard_screen_scaling, 288
  screen_count, 288
  screen_dpi, 288
  screen_num, 289
  screen_scale, 289
  screen_scaling_supported, 289
  screen_work_area, 290
  screen_xxywh, 291, 292

screen_count
  Screen functions, 288

screen_dpi
  Screen functions, 288

screen_num
  Fl_Window, 1934
  Screen functions, 289

screen_scale
  Screen functions, 289

screen_scaling_supported
  Screen functions, 289

screen_work_area
  Screen functions, 290

screen_xxywh
  Screen functions, 291, 292

scroll
  Fl_Terminal, 1597
  Fl_Text_Display, 1659

scroll_
  Fl_Text_Display, 1659

scroll_timer_cb
  Fl_Text_Display, 1659

scroll_to
  Fl_Scroll, 1365

scrollbar
  Fl_Browser_, 517
  Fl_Terminal, 1604

scrollbar_align
  Fl_Text_Display, 1659

scrollbar_left
  Fl_Browser_, 514

scrollbar_right
  Fl_Browser_, 515

scrollbar_size
  Fl, 415, 416
  Fl_Browser_, 515
  Fl_Help_View, 879, 880
  Fl_Scroll, 1365
  Fl_Table, 1526
  Fl_Terminal, 1598
  Fl_Text_Display, 1660
  Fl_Tree, 1787

scrollbar_width
  Fl_Browser_, 515
  Fl_Text_Display, 1660

scrollvalue
  Fl_Slider, 1451

search_backward
  Fl_Text_Buffer, 1619

search_forward
  Fl_Text_Buffer, 1619

second
  Fl_Clock_Output, 609

secondary_selection_color
  Fl_Text_Display, 1660, 1661

secondary_selection_text
  Fl_Text_Buffer, 1620

seconds_between
  Fl, 416

seconds_since
  Fl, 416

select
  Fl_Browser, 490
  Fl_Browser_, 515
  Fl_Tree, 1787, 1788
  Fl_Tree_Item, 1812

select_all
  Fl_Tree, 1788
  Fl_Tree_Item, 1812

select_all_rows
  Fl_Table_Row, 1543

select_only
  Fl_Browser_, 516
  Fl_Tree, 1789

select_row
  Fl_Table_Row, 1543

select_toggle
  Fl_Tree, 1789

selectbox
  Fl_Tree, 1790

selected
  Fl_Browser, 490
  Fl_Text_Selection, 1694

Generated by Doxygen
selection
  FL_Browser_, 516
  Selection & Clipboard functions, 285
Selection & Clipboard functions, 282
  add_clipboard_notify, 283
  clipboard_contains, 283
  copy, 284
  dnd, 284
  paste, 285
  selection, 285
  selection_owner, 286
  selection_to_clipboard, 286
selection_color
  FL_Widget, 1897
selection_extend
  FL_Terminal, 1598
selection_owner
  Selection & Clipboard functions, 286
selection_text
  FL_Terminal, 1598
  FL_Text_Buffer, 1620
selection_text_len
  FL_Terminal, 1598
selection_to_clipboard
  Selection & Clipboard functions, 286
selectmode
  FL_Tree, 1790
  FL_Tree_Prefs, 1822
set
  FL_Button, 528
  FL_FormsBitmap, 757
  FL_FormsPixmap, 764
  FL_Menu_Item, 1087
  FL_Preferences, 1244, 1245, 1247
  FL_Text_Selection, 1695
set_active
  FL_Widget, 1897
set_atclose
  Windows handling functions, 268
set_box_color
  FL, 416
set_changed
  FL_Widget, 1898
set_color
  Color & Font functions, 305
set_current
  FL_Copy_Surface, 627
  FL_Image_Surface, 947
  FL_PostScript_File_Device, 1227
  FL_Printer, 1256
  FL_Surface_Device, 1476
set_cursor
  FL_Tile, 1712
set_draw_cb
  FL_Cairo_Window, 543
set_font
  Color & Font functions, 305
set_fonts
  Color & Font functions, 306
set_idle
  FL, 417
set_item_focus
  FL_Tree, 1790
set_menu_window
  FL_Window, 1934
set_modal
  FL_Window, 1935
set_non_modal
  FL_Window, 1935
set_output
  FL_Widget, 1898
set_selection
  FL_Table, 1526
set_tooltip_window
  FL_Window, 1935
set_visible
  FL_Widget, 1898
set_visible_focus
  FL_Widget, 1898
setonly
  FL_Menu_Item, 1087
shadow
  FL_Menu_Item, 1087
shape
  FL_Clock_Output, 610
shortcut
  FL_Button, 528
  FL_Input_, 979
  FL_Menu_Item, 1088
  FL_Text_Display, 1661
  FL_Value_Input, 1843
SHORTCUT_LABEL
  FL_Widget, 1872
shortcut_label
  FL_Widget, 1898
show
  FL_Browser, 491
  FL_Double_Window, 663
  FL_Gl_Window, 804
  FL_Help_Dialog, 864
  FL_Native_File_Chooser, 1143
  FL_Overlay_Window, 1176
  FL_Single_Window, 1441
  FL_Widget, 1898
  FL_Window, 1936
show_cursor
  FL_Text_Display, 1661
show_grid
  FL_Grid, 841, 842
show_insert_position
  FL_Text_Display, 1661
show_item
  FL_Tree, 1790, 1791
show_item_bottom
  FL_Tree, 1791
show_item_middle

Generated by Doxygen
INDEX

FL_Tree, 1791
show_item_top
FL_Tree, 1791
show_next_window_iconic
FL_Window, 1937
show_self
FL_Tree, 1792
FL_Tree_Item, 1812
show_unknown
FL_Terminal, 1599
show_widgets
FL_Tree_Item, 1812
showCollapse
FL_Tree, 1792
FL_Tree_Prefs, 1822
showHiddenButton
FL_File_Chooser, 689
shown
FL_File_Chooser, 689
FL_Window, 1937
showroot
FL_Tree, 1792
FL_Tree_Prefs, 1822
SIMPLE_CURSOR
FL_Text_Display, 1639
SINGLE
FL_File_Chooser, 687
size
FL_Browser, 491
FL_Calc, 556
FL_Input_, 980
FL_Menu_, 1052
FL_Menu_Item, 1088
FL_Preferences, 1248
FL_Tooltip, 1744
FL_Widget, 1899
size_range
FL_Tile, 1712
FL_Window, 1937
sizes
FL_Group, 861
skip_displayed_characters
FL_Text_Buffer, 1620
skip_lines
FL_Text_Display, 1662
slider_size
FL_Slider, 1451
slowarrow.h, 2292
soft
FL_Adjuster, 429
FL_Value_Input, 1844
FL_Value_Output, 1854
Software License, 181
sort
FL_Browser_. 516
sortorder
FL_Tree, 1792
FL_Tree_Prefs, 1822
spacing
FL_Flex, 740
spacing.h, 2323
speed
FL_Anim_GIF_Image, 443
spelling_underline_color
FL_Text_Display, 1662
start
FL_Anim_GIF_Image, 444
FL_Text_Selection, 1695
start_job
FL_Paged_Device, 1192
FL_PostScript_File_Device, 1227
start_page
FL_Paged_Device, 1192
static_value
FL_Input_, 980
step
FL_Counter, 638
FL_Spinner, 1462
FL_Valuator, 1833
stop
FL_Anim_GIF_Image, 444
STRICT_RFC3629
Unicode and UTF-8 functions, 347
STRIKEOUT
FL_Terminal, 1586
String handling functions, 361
fl_strdup, 361
string_width
FL_Text_Display, 1662
style_buffer
FL_Text_Display, 1663
submenu
FL_Menu_Item, 1088
surface
FL_Surface_Device, 1476
suspended
FL_Timer, 1731
swap
FL_Browser, 491, 492
swap_buffers
FL_GL_Window, 804
swap_children
FL_Tree_Item, 1813
swap_interval
FL_GL_Window, 804, 805
swapping
FL_Browser_, 516
symbol_.h, 2345
SYSTEM
FL_Preferences, 1232
SYSTEM_L
FL_Preferences, 1233
tab_align
FL_Tabs, 1562
tab_cell-nav
FL_Table, 1527
update
Fl_Menu_Bar, 1064
Fl_Shared_Image, 1412
Fl_Sys_Menu_Bar, 1502
Fl_Text_Selection, 1695
update_child
Fl_Group, 862
update_h_scrollbar
Fl_Text_Display, 1664
update_line_starts
Fl_Text_Display, 1664
update_menubutton
Fl_Input_Choice, 998
update_prev_next
Fl_Tree_Item, 1813
update_vScrollbar
Fl_Text_Display, 1664
USE_FILTER_EXT
Fl_Native_File_Chooser, 1140
use_high_res_GL
FI, 418
USER
Fl_Preferences, 1232
user_data
Fl_Widget, 1902
USER_L
Fl_Preferences, 1233
userdeicon
Fl_Tree_Item, 1814
Fl_Tree_Prefs, 1822
USERFLAG1
Fl_Widget, 1873
USERFLAG2
Fl_Widget, 1873
USERFLAG3
Fl_Widget, 1873
usericon
Fl_Tree, 1792, 1793
Fl_Tree_Item, 1814
Using OpenGL, 88
utf8.h, 2638
utf8_internal.h, 2292
utf8Utils.c, 2649
valid
Fl_Anim_GIF_Image, 444
Fl_GL_Window, 805
value
Fl_Browser, 493
Fl_Button, 529
Fl_Choice, 591, 592
Fl_Clock_Output, 610, 611
Fl_Color_Chooser, 623
Fl_File_Chooser, 689
Fl_File_Input, 705, 706
Fl_Help_Dialog, 864, 865
Fl_Help_View, 880
Fl_Input, 984, 985
Fl_Input_Choice, 999
Fl_Menu, 1053
Fl_Menu_Item, 1089
Fl_Progress, 1264
Fl_Scrollbar, 1376
Fl_Shortcut_Button, 1420
Fl_Spinner, 1463
Fl_Tabs, 1563, 1564
Fl_Valuator, 1833
value_damage
Fl_Adjuster, 429
Fl_Valuator, 1833
value_height
Fl_Value_Slider, 1864
value_width
Fl_Value_Slider, 1865
version
Fl, 418
VERTICAL
Fl_Browser_, 505
Fl_Flex, 733
VERTICAL_ALWAYS
Fl_Browser_, 505
viscii.h, 2639
visible
Fl_Browser, 493
Fl_Widget, 1902
visible_cells
Fl_Table, 1528
VISIBLE_FOCUS
Fl_Widget, 1873
visible_focus
Fl, 418
Fl_Widget, 1902
visible_r
Fl_Tree_Item, 1814
Fl_Widget, 1902
visual
Fl, 418
vline_length
Fl_Text_Display, 1665
vposition
Fl_Browser_, 517
Fl_Tree, 1793
vprintf
Fl_Terminal, 1603
Fl_Text_Buffer, 1621
vsnprintf.c, 2293
fl_vsnprintf, 2293
w
Fl_Image, 941
Fl_Widget, 1903
w_to_col
Fl_Terminal, 1603
wait
Fl, 419
wait_for_expose
Fl_Window, 1938
walk_selection